

axiomTM



The 30 Year Horizon

| | | |
|------------------------------|------------------------|---------------------------|
| <i>Manuel Bronstein</i> | <i>William Burge</i> | <i>Timothy Daly</i> |
| <i>James Davenport</i> | <i>Michael Dewar</i> | <i>Martin Dunstan</i> |
| <i>Albrecht Fortenbacher</i> | <i>Patrizia Gianni</i> | <i>Johannes Grabmeier</i> |
| <i>Jocelyn Guidry</i> | <i>Richard Jenks</i> | <i>Larry Lambe</i> |
| <i>Michael Monagan</i> | <i>Scott Morrison</i> | <i>William Sit</i> |
| <i>Jonathan Steinbach</i> | <i>Robert Sutor</i> | <i>Barry Trager</i> |
| <i>Stephen Watt</i> | <i>Jim Wen</i> | <i>Clifton Williamson</i> |

Volume 10: Axiom Algebra: Numerics

Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan

Portions Copyright (c) 2007 Alfredo Portes

Portions Copyright (c) 2007 Arthur Ralfs

Portions Copyright (c) 2005 Timothy Daly

Portions Copyright (c) 1991-2002,
The Numerical ALgorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical ALgorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

| | | |
|-----------------------|-----------------------|------------------------|
| Michael Albaugh | Cyril Alberga | Roy Adler |
| Christian Aistleitner | Richard Anderson | George Andrews |
| S.J. Atkins | Henry Baker | Martin Baker |
| Stephen Balzac | Yurij Baransky | David R. Barton |
| Gerald Baumgartner | Gilbert Baumslag | Michael Becker |
| Nelson H. F. Beebe | Jay Belanger | David Bindel |
| Fred Blair | Vladimir Bondarenko | Mark Botch |
| Raoul Bourquin | Alexandre Bouyer | Karen Braman |
| Peter A. Broadbery | Martin Brock | Manuel Bronstein |
| Stephen Buchwald | Florian Bundschuh | Luanne Burns |
| William Burge | Ralph Byers | Quentin Carpent |
| Robert Caviness | Bruce Char | Ondrej Certik |
| Tzu-Yi Chen | Cheekai Chin | David V. Chudnovsky |
| Gregory V. Chudnovsky | Mark Clements | James Cloos |
| Jia Zhao Cong | Josh Cohen | Christophe Conil |
| Don Coppersmith | George Corliss | Robert Corless |
| Gary Cornell | Meino Cramer | Jeremy Du Croz |
| David Cyganski | Nathaniel Daly | Timothy Daly Sr. |
| Timothy Daly Jr. | James H. Davenport | David Day |
| James Demmel | Didier Deshommies | Michael Dewar |
| Jack Dongarra | Jean Della Dora | Gabriel Dos Reis |
| Claire DiCrescendo | Sam Dooley | Lionel Ducos |
| Iain Duff | Lee Duhem | Martin Dunstan |
| Brian Dupee | Dominique Duval | Robert Edwards |
| Heow Eide-Goodman | Lars Erickson | Richard Fateman |
| Bertfried Fauser | Stuart Feldman | John Fletcher |
| Brian Ford | Albrecht Fortenbacher | George Frances |
| Constantine Frangos | Timothy Freeman | Korrinn Fu |
| Marc Gaetano | Rudiger Gebauer | Van de Geijn |
| Kathy Gerber | Patricia Gianni | Samantha Goldrich |
| Holger Gollan | Teresa Gomez-Diaz | Laureano Gonzalez-Vega |
| Stephen Gortler | Johannes Grabmeier | Matt Grayson |
| Klaus Ebbe Grue | James Griesmer | Vladimir Grinberg |
| Oswald Gschnitzer | Ming Gu | Jocelyn Guidry |
| Gaetan Hache | Steve Hague | Satoshi Hamaguchi |
| Sven Hammarling | Mike Hansen | Richard Hanson |
| Richard Harke | Bill Hart | Vilya Harvey |
| Martin Hassner | Arthur S. Hathaway | Dan Hatton |
| Waldek Hebisch | Karl Hegbloom | Ralf Hemmecke |

| | | |
|----------------------|------------------------|------------------------|
| Henderson | Antoine Hersen | Roger House |
| Gernot Hueber | Pietro Iglio | Alejandro Jakubi |
| Richard Jenks | William Kahan | Kai Kaminski |
| Grant Keady | Wilfrid Kendall | Tony Kennedy |
| Ted Kosan | Paul Kosinski | Klaus Kusche |
| Bernhard Kutzler | Tim Lahey | Larry Lambe |
| Kaj Laurson | George L. Legendre | Franz Lehner |
| Frederic Lehouby | Michel Levaud | Howard Levy |
| Ren-Cang Li | Rudiger Loos | Michael Lucks |
| Richard Luczak | Camm Maguire | Francois Maltey |
| Alasdair McAndrew | Bob McElrath | Michael McGettrick |
| Edi Meier | Ian Meikle | David Mentre |
| Victor S. Miller | Gerard Milmeister | Mohammed Mobarak |
| H. Michael Moeller | Michael Monagan | Marc Moreno-Maza |
| Scott Morrison | Joel Moses | Mark Murray |
| William Naylor | Patrice Naudin | C. Andrew Neff |
| John Nelder | Godfrey Nolan | Arthur Norman |
| Jinzhong Niu | Michael O'Connor | Summat Oemrawsingh |
| Kostas Oikonomou | Humberto Ortiz-Zuazaga | Julian A. Padget |
| Bill Page | David Parnas | Susan Pelzel |
| Michel Petitot | Didier Pinchon | Ayal Pinkus |
| Frederick H. Pitts | Jose Alfredo Portes | Gregorio Quintana-Orti |
| Claude Quitte | Arthur C. Ralfs | Norman Ramsey |
| Anatoly Raportirenko | Albert D. Rich | Michael Richardson |
| Guilherme Reis | Huan Ren | Renaud Rioboo |
| Jean Rivlin | Nicolas Robidoux | Simon Robinson |
| Raymond Rogers | Michael Rothstein | Martin Rubey |
| Philip Santas | Alfred Scheerhorn | William Schelter |
| Gerhard Schneider | Martin Schoenert | Marshall Schor |
| Frithjof Schulze | Fritz Schwarz | Steven Segletes |
| V. Sima | Nick Simicich | William Sit |
| Elena Smirnova | Jonathan Steinbach | Fabio Stumbo |
| Christine Sundaresan | Robert Sutor | Moss E. Sweedler |
| Eugene Surowitz | Max Tegmark | T. Doug Telford |
| James Thatcher | Balbir Thomas | Mike Thomas |
| Dylan Thurston | Steve Toleque | Barry Trager |
| Themos T. Tsikas | Gregory Vanuxem | Bernhard Wall |
| Stephen Watt | Jaap Weel | Juergen Weiss |
| M. Weller | Mark Wegman | James Wen |
| Thorsten Werther | Michael Wester | R. Clint Whaley |
| John M. Wiley | Berhard Will | Clifton J. Williamson |
| Stephen Wilson | Shmuel Winograd | Robert Wisbauer |
| Sandra Wityak | Waldemar Wiwianka | Knut Wolf |
| Liu Xiaojun | Clifford Yapp | David Yun |
| Vadim Zhytnikov | Richard Zippel | Evelyn Zoernack |
| Bruno Zuercher | Dan Zwillinger | |

Contents

| | | |
|----------|--------------------------------------|----------|
| 1 | Numerical Analysis [4] | 1 |
| 2 | Chapter Overview | 3 |
| 3 | Algebra Cover Code | 7 |
| | package BLAS1 BlasLevelOne | 7 |
| | BlasLevelOne (BLAS1) | 42 |
| | dcabs1 BLAS | 48 |
| | lsame BLAS | 53 |
| | daxpy BLAS | 69 |
| | dcopy BLAS | 81 |
| | ddot BLAS | 91 |
| | dnrm2 BLAS | 99 |
| | drotg BLAS | 105 |
| | drot BLAS | 117 |
| | dscal BLAS | 134 |
| | dswap BLAS | 141 |
| | dzasum BLAS | 149 |
| | dznrm2 BLAS | 154 |
| | icamax BLAS | 162 |
| | idamax BLAS | 170 |
| | isamax BLAS | 177 |
| | izamax BLAS | 185 |
| | zaxpy BLAS | 192 |
| | zcopy BLAS | 208 |
| | zdotc BLAS | 212 |
| | zdotu BLAS | 216 |
| | zdscal BLAS | 219 |
| | zrotg BLAS | 222 |
| | zscal BLAS | 226 |
| | zswap BLAS | 229 |

| | | |
|----------|---------------------|------------|
| 4 | BLAS Level 2 | 235 |
| | dgbmv BLAS | 235 |
| | dgemv BLAS | 248 |
| | dger BLAS | 258 |
| | dsbmv BLAS | 265 |
| | dspmv BLAS | 278 |
| | dspr2 BLAS | 290 |
| | dspr BLAS | 301 |
| | dsymv BLAS | 310 |
| | dsyr2 BLAS | 322 |
| | dsyr BLAS | 333 |
| | dtbmv BLAS | 341 |
| | dtbsv BLAS | 358 |
| | dtpmv BLAS | 375 |
| | dtpsv BLAS | 391 |
| | dtrmv BLAS | 407 |
| | dtrsv BLAS | 421 |
| | zgbmv BLAS | 435 |
| | zgemv BLAS | 450 |
| | zgerc BLAS | 462 |
| | zgeru BLAS | 468 |
| | zhbmv BLAS | 474 |
| | zhemv BLAS | 488 |
| | zher2 BLAS | 500 |
| | zher BLAS | 515 |
| | zhpmv BLAS | 526 |
| | zhpr2 BLAS | 539 |
| | zhpr BLAS | 554 |
| | ztbmv BLAS | 566 |
| | ztbsv BLAS | 586 |
| | ztpmv BLAS | 607 |
| | ztpsv BLAS | 626 |
| | ztrmv BLAS | 646 |
| | ztrsv BLAS | 664 |
| 5 | BLAS Level 3 | 683 |
| | dgemm BLAS | 683 |
| | dsymm BLAS | 696 |
| | dsyr2k BLAS | 710 |
| | dsyrk BLAS | 725 |
| | dtrmm BLAS | 739 |
| | dtrsm BLAS | 757 |
| | zgemm BLAS | 776 |
| | zhemm BLAS | 796 |
| | zher2k BLAS | 811 |
| | zherk BLAS | 831 |

| | |
|---------------------------|------------|
| zsymm BLAS | 849 |
| zsyr2k BLAS | 863 |
| zsyrk BLAS | 878 |
| ztrmm BLAS | 891 |
| ztrsm BLAS | 912 |
| 6 LAPACK | 937 |
| dbdsdc LAPACK | 937 |
| dbdsqr LAPACK | 957 |
| ddisna LAPACK | 995 |
| dgebak LAPACK | 1003 |
| dgebal LAPACK | 1010 |
| dgebd2 LAPACK | 1022 |
| dgebrd LAPACK | 1032 |
| dgeev LAPACK | 1042 |
| dgeevx LAPACK | 1062 |
| dgehd2 LAPACK | 1086 |
| dgehrd LAPACK | 1092 |
| dgelq2 LAPACK | 1102 |
| dgelqf LAPACK | 1106 |
| dgeqr2 LAPACK | 1114 |
| dgeqrf LAPACK | 1118 |
| dgesdd LAPACK | 1125 |
| dgesvd LAPACK | 1193 |
| dgesv LAPACK | 1391 |
| dgetf2 LAPACK | 1395 |
| dgetrf LAPACK | 1400 |
| dgetrs LAPACK | 1407 |
| dhseqr LAPACK | 1412 |
| disnan LAPACK | 1432 |
| dlabad LAPACK | 1434 |
| dlabrd LAPACK | 1436 |
| dlacon LAPACK | 1454 |
| dlacpy LAPACK | 1462 |
| dladiv LAPACK | 1466 |
| dlaed6 LAPACK | 1468 |
| dlaexc LAPACK | 1481 |
| dlahqr LAPACK | 1499 |
| dlahrd LAPACK | 1522 |
| dlaisnan LAPACK | 1531 |
| dlaln2 LAPACK | 1533 |
| dlamch LAPACK | 1559 |
| dlamc1 LAPACK | 1564 |
| dlamc2 LAPACK | 1571 |
| dlamc3 LAPACK | 1582 |
| dlamc4 LAPACK | 1584 |

| | |
|-------------------------|------|
| dlamc5 LAPACK | 1588 |
| dlamrg LAPACK | 1594 |
| dlange LAPACK | 1599 |
| dlanhs LAPACK | 1605 |
| dlanst LAPACK | 1611 |
| dlanv2 LAPACK | 1616 |
| dlapy2 LAPACK | 1624 |
| dlapy3 LAPACK | 1626 |
| dlaqtr LAPACK | 1629 |
| dlarfb LAPACK | 1666 |
| dlarfg LAPACK | 1691 |
| dlarf LAPACK | 1696 |
| dlarft LAPACK | 1700 |
| dlarfx LAPACK | 1710 |
| dlartg LAPACK | 1765 |
| dlas2 LAPACK | 1771 |
| dlascl LAPACK | 1775 |
| dlasd0 LAPACK | 1787 |
| dlasd1 LAPACK | 1798 |
| dlasd2 LAPACK | 1806 |
| dlasd3 LAPACK | 1827 |
| dlasd4 LAPACK | 1846 |
| dlasd5 LAPACK | 1895 |
| dlasd6 LAPACK | 1904 |
| dlasd7 LAPACK | 1914 |
| dlasd8 LAPACK | 1932 |
| dlasda LAPACK | 1945 |
| dlasdq LAPACK | 1964 |
| dlasdt LAPACK | 1977 |
| dlaset LAPACK | 1983 |
| dlasq1 LAPACK | 1987 |
| dlasq2 LAPACK | 1994 |
| dlasq3 LAPACK | 2021 |
| dlasq4 LAPACK | 2040 |
| dlasq5 LAPACK | 2058 |
| dlasq6 LAPACK | 2072 |
| dlasr LAPACK | 2084 |
| dlasrt LAPACK | 2103 |
| dlasq LAPACK | 2113 |
| dlasv2 LAPACK | 2117 |
| dlaswp LAPACK | 2125 |
| dlasy2 LAPACK | 2131 |
| dorg2r LAPACK | 2154 |
| dorgbr LAPACK | 2159 |
| dorghr LAPACK | 2169 |
| dorgl2 LAPACK | 2175 |

| | |
|---------------|------|
| dorglq LAPACK | 2181 |
| dorgqr LAPACK | 2189 |
| dorm2r LAPACK | 2198 |
| dormbr LAPACK | 2205 |
| dorml2 LAPACK | 2215 |
| dormlq LAPACK | 2222 |
| dormqr LAPACK | 2232 |
| dtrevc LAPACK | 2242 |
| dtrexc LAPACK | 2302 |
| dtrsna LAPACK | 2317 |
| ieeck LAPACK | 2340 |
| ilaenv LAPACK | 2346 |
| ilazlc LAPACK | 2367 |
| ilazlr LAPACK | 2370 |
| zgebak LAPACK | 2374 |
| zgebal LAPACK | 2383 |
| zgeev LAPACK | 2397 |
| zgehd2 LAPACK | 2417 |
| zgehrd LAPACK | 2424 |
| zhseqr LAPACK | 2436 |
| zlacgv LAPACK | 2451 |
| zlacpy LAPACK | 2455 |
| zladv LAPACK | 2459 |
| zlahqr LAPACK | 2462 |
| zlahr2 LAPACK | 2485 |
| zlange LAPACK | 2499 |
| zlaqr0 LAPACK | 2505 |
| zlaqr1 LAPACK | 2531 |
| zlaqr2 LAPACK | 2537 |
| zlaqr3 LAPACK | 2557 |
| zlaqr4 LAPACK | 2579 |
| zlaqr5 LAPACK | 2604 |
| zlarfb LAPACK | 2654 |
| zlarf LAPACK | 2689 |
| zlarfg LAPACK | 2696 |
| zlarft LAPACK | 2701 |
| zlartg LAPACK | 2715 |
| zlascl LAPACK | 2723 |
| zlaset LAPACK | 2735 |
| zlassq LAPACK | 2740 |
| zlatrs LAPACK | 2745 |
| zrot LAPACK | 2779 |
| ztrevc LAPACK | 2784 |
| ztrexc LAPACK | 2803 |
| zung2r LAPACK | 2810 |
| zunghr LAPACK | 2816 |

| | |
|----------------------------|-------------|
| zungqr LAPACK | 2824 |
| zunm2r LAPACK | 2834 |
| zunmhr LAPACK | 2842 |
| zunmqr LAPACK | 2850 |
| 7 LAPACK tests | 2863 |
| 8 Chunk collections | 2879 |
| 9 Index | 2891 |

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

Chapter 1

Numerical Analysis [4]

We can describe each number as x^* which has a machine-representable form which differs from the number x it is intended to represent. Quoting Householder we get:

$$x^* = \pm(x_1\beta^{-1} + x_2\beta^{-2} + \cdots + x_\lambda\beta^\lambda)\beta^\sigma$$

where β is the base, usually 2 or 10, λ is a positive integer, and σ is any integer, possibly zero. It may be that λ is fixed throughout the course of the computation, or it may vary, but in any case it is limited by practical considerations. Such a number will be called a representation. It may be that x^* is obtained by “rounding off” a number whose true value is x (for example, $x = 1/3$, $x^* = 0.33$), or that x^* is the result of measuring physically a quantity whose true value is x , or that x^* is the result of a computation intended to give an approximation to the quantity x .

Suppose one is interested in performing an operations ω upon a pair of numbers x and y . That is to say, $x\omega y$ may represent a product of x and y , a quotient of x by y , the y th power of x , In the numerical computation, however, one has only x^* and y^* upon which to operate, not x and y (or at least these are the quantities upon which one does, in fact operate). Not only this, but often one does not even perform the strict operation ω , but rather a pseudo operation ω^* , which yields a rounded-off product, quotient, power, etc. Hence, instead of obtaining the desired result $x\omega y$, one obtains a result $x^*\omega^*y^*$.

The error in the result is therefore

$$x\omega y - x^*\omega^*y^* = (x\omega y - x^*\omega y^*) + (x^*\omega y^* - x^*\omega^*y^*)$$

Since x^* and y^* are numbers, the operation ω can be applied to them, and $x^*\omega y^*$ is well defined, except for special cases as when ω represents division and $y^* = 0$. But the expression in the first parentheses on the right represents propagated error, and that in the second parentheses represents generated error, or round-off. Hence the total error in the result is the sum of the error propagated by the operation and that generated by the operation.

Householder notes that, given two operations ω and ϕ , it may be true that the operations are associative, e.g:

$$(x^*\omega y^*)\phi z^* = x^*\omega(y^*\phi z^*)$$

but if we expand these in terms of the above definitions of propagation and generation error we get two different expressions:

$$\begin{aligned} (x^*\omega y^*)\phi z^* - (x^*\omega^* y^*)\phi^* z^* &= [(x^*\omega y^*)\phi z^* - (x^*\omega^* y^*)\phi z^*] \\ &+ [(x^*\omega^* y^*)\phi z^* - (x^*\omega^* y^*)\phi^* z^*] \end{aligned}$$

$$\begin{aligned} x^*\omega(y^*\phi z^*) - x^*\omega^*(y^*\phi^* z^*) &= [x^*\omega(y^*\phi z^*) - x^*\omega(y^*\phi^* z^*)] \\ &+ [x^*\omega(y^*\phi^* z^*) - x^*\omega^*(y^*\phi^* z^*)] \end{aligned}$$

These are not always equal which implies that the strictly machine operations are not necessarily commutative.

Householder distinguishes a third class of error (besides propagation and generative) called residual errors. This occurs because some functions are approximated by infinite series. The finite computation of the series forces the truncation of the remaining terms causing these residual errors.

We will try to perform an analysis of each of the routines in this library for the given inputs to try to see the propagation errors and generation errors they introduce. Every effort will be made to minimize these errors. In particular, we will appeal to the machine generated code to see what approximations actually occur.

Chapter 2

Chapter Overview

Each routine in the Basic Linear Algebra Subroutine set (BLAS) has a prefix where:

- C - Complex
- D - Double Precision
- S - Real
- Z - Complex*16

Routines in level 2 and level 3 of BLAS use the prefix for type:

- GE - general
- GB - general band
- SY - symmetric
- HE - hermitian
- TR - triangular
- SB - symmetric band
- HB - hermetian band
- TB - triangular band
- SP - Sum packed
- HP - hermitian packed
- TP - triangular packed

For level 2 and level 3 BLAS options the options argument is CHARACTER*1 and may be passed as character strings. They mean:

- TRANx
 - No transpose
 - Transpose
 - Conjugate transpose (X , X^T , X^H)
- UPLO
 - Upper triangular
 - Lower triangular
- DIAG
 - Non-unit triangular
 - Unit triangular
- SIDE
 - Left - A or op(A) on the left
 - Right - A or op(A) on the right

For real matrices, TRANSx=T and TRANSx=C have the same meaning. For Hermitian matrices, TRANSx=T is not allowed. For complex symmetric matrices, TRANSx=H is not allowed.

There were 38 BLAS Level 1 routines defined in [?]. They are

- Dot product SDSDOT, DSDOT, DQ-IDOT DQ-ADOT C-UDOT C-CDOT DDOT SDOT
- Constant times a vector plus a vector CAXPY DAXPY SAXPY
- Set up Givens rotation DROTG SROTG
- Apply rotation DROT SROT
- Set up modified Givens rotation DROTMG SROTMG
- Apply modified rotation DROTM SROTM
- Copy x into y CCOPY DCOPY SCOPY
- Swap x and y CSWAP DSWAP SSWAP
- 2-norm (Euclidean length) SCNRM2 DNRM2 SNRM2
- Sum of absolute values SCASUM DASUM SASUM

- Constant times a vector CSSCAL CSCAL DSCAL SSCAL
- Index of maximum element ICAMAX IDAMAX ISAMAX

where

- I Integer
- S Single Precision
- D Double Precision
- C Single Precision Complex
- Q extended precision
- Z COMPLEX*16

Vector arguments are permitted to have a storage spacing between elements. This spacing is specified by an increment parameter. For example, suppose a vector x having components x_i , $i = 1, \dots, N$ is stored in a DOUBLE PRECISION array $DX()$ with increment parameter $INCX$. If $INCX \geq 0$ then x_i is stored in $DX(1 + (i - 1) * INCX)$. If $INCX < 0$ then x_i is stored in $DX(1 + (N - 1) * |INCX|)$.

Chapter 3

Algebra Cover Code

package BLAS1 BlasLevelOne

— BlasLevelOne.input —

```
)set break resume
)sys rm -f BlasLevelOne.output
)spool BlasLevelOne.output
)set message test on
)set message auto off
)clear all
```

--S 1 of 208

```
t1:Complex DoubleFloat := complex(1.0,0)
```

--R

--R

--R (1) 1.

--R

Type: Complex(DoubleFloat)

--E 1

--S 2 of 208

```
dcabs1(t1)
```

--R

--R

--R (2) 1.

--R

Type: DoubleFloat

--E 2

--S 3 of 208

```
t2:Complex DoubleFloat := complex(1.0,1.0)
```

--R

--R

[illegible]

```

--E 9

--S 10 of 208
dcabs1(t5)
--R
--R
--R (10) 4.
--R
--R                                          Type: DoubleFloat
--E 10

)clear all

--S 11 of 208
a:PRIMARR(DFLOAT):=[ [1.0,2.0,3.0,4.0,5.0,6.0] ]
--R
--R (1) [1.,2.,3.,4.,0.,5.,0.,6.,0.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 11

--S 12 of 208
dasum(3,a,-1) -- 0.0 neg incx
--R
--R (2) 0.
--R
--R                                          Type: DoubleFloat
--E 12

--S 13 of 208
dasum(3,a,0) -- 0.0 zero incx
--R
--R (3) 0.
--R
--R                                          Type: DoubleFloat
--E 13

--S 14 of 208
dasum(-1,a,1) -- 0.0 neg elements
--R
--R (4) 0.
--R
--R                                          Type: DoubleFloat
--E 14

--S 15 of 208
dasum(0,a,1) -- 0.0 no elements
--R
--R (5) 0.
--R
--R                                          Type: DoubleFloat
--E 15

--S 16 of 208
dasum(1,a,1) -- 1.0 1.0
--R

```

```

--R (6) 1.
--R
--R                                          Type: DoubleFloat
--E 16

--S 17 of 208
dasum(2,a,1) -- 3.0 1.0+2.0
--R
--R (7) 3.
--R
--R                                          Type: DoubleFloat
--E 17

--S 18 of 208
dasum(3,a,1) -- 6.0 1.0+2.0+3.0
--R
--R (8) 6.
--R
--R                                          Type: DoubleFloat
--E 18

--S 19 of 208
dasum(4,a,1) -- 10.0 1.0+2.0+3.0+4.0
--R
--R (9) 10.
--R
--R                                          Type: DoubleFloat
--E 19

--S 20 of 208
dasum(5,a,1) -- 15.0 1.0+2.0+3.0+4.0+5.0
--R
--R (10) 10.
--R
--R                                          Type: DoubleFloat
--E 20

--S 21 of 208
dasum(6,a,1) -- 21.0 1.0+2.0+3.0+4.0+5.0+6.0
--R
--R (11) 15.
--R
--R                                          Type: DoubleFloat
--E 21

--S 22 of 208
dasum(7,a,1) -- 21.0 1.0+2.0+3.0+4.0+5.0+6.0
--R
--R (12) 15.
--R
--R                                          Type: DoubleFloat
--E 22

--S 23 of 208
dasum(1,a,2) -- 1.0 1.0
--R
--R (13) 1.

```

[illegible]

--E 30

--S 31 of 208

dasum(2,a,4) -- 6.0 1.0+5.0

--R

--R (21) 1.

--R

Type: DoubleFloat

--E 31

--S 32 of 208

dasum(3,a,4) -- 6.0 1.0+5.0

--R

--R (22) 1.

--R

Type: DoubleFloat

--E 32

--S 33 of 208

dasum(1,a,5) -- 1.0 1.0

--R

--R (23) 1.

--R

Type: DoubleFloat

--E 33

--S 34 of 208

dasum(2,a,5) -- 7.0 1.0+6.0

--R

--R (24) 6.

--R

Type: DoubleFloat

--E 34

--S 35 of 208

dasum(3,a,5) -- 7.0 1.0+6.0

--R

--R (25) 6.

--R

Type: DoubleFloat

--E 35

--S 36 of 208

dasum(1,a,6) -- 1.0 1.0

--R

--R (26) 1.

--R

Type: DoubleFloat

--E 36

--S 37 of 208

dasum(2,a,6) -- 1.0 1.0

--R

--R (27) 1.

--R

Type: DoubleFloat

--E 37


```

--S 38 of 208
dasum(1,a,7) -- 1.0 1.0
--R
--R (28) 1.
--R
--R                                          Type: DoubleFloat
--E 38

)clear all

--S 39 of 208
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (1) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 39

--S 40 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (2) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 40

--S 41 of 208
daxpy(3,2.0,a,1,b,1)
--R
--R
--R (3) [3.,6.,9.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 41

--S 42 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (4) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 42

--S 43 of 208
daxpy(7,2.0,a,1,b,1)
--R
--R
--R (5) [3.,6.,9.,12.,15.,18.,21.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 43

```

```

--S 44 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (6) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 44

```

Note that Axiom properly handles array indexes that are out of bounds. The BLAS daxpy routine cannot check this condition.

— **BlasLevelOne.input** —

```

--S 45 of 208
daxpy(8,2.0,a,1,b,1)
--R
--R
--R (7) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 45

```

```

--S 46 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (8) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 46

```

```

--S 47 of 208
daxpy(3,2.0,a,3,b,3)
--R
--R
--R (9) [3.,2.,3.,12.,5.,6.,21.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 47

```

```

--S 48 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (10) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 48

```

```

--S 49 of 208
daxpy(4,2.0,a,2,b,2)
--R

```

```

--R
--R (11) [3.,2.,9.,4.,15.,6.,21.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 49

--S 50 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (12) [1.,2.,3.,4.,5.,6.,7.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 50

--S 51 of 208
daxpy(5,2.0,a,2,b,2)
--R
--R
--R (13) [1.,2.,3.,4.,5.,6.,7.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 51

--S 52 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (14) [1.,2.,3.,4.,5.,6.,7.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 52

--S 53 of 208
daxpy(3,2.0,a,2,b,2)
--R
--R
--R (15) [3.,2.,9.,4.,15.,6.,7.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 53

--S 54 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (16) [1.,2.,3.,4.,5.,6.,7.]
--R                                         Type: PrIMITIVEArray(DoubleFloat)
--E 54

--S 55 of 208
daxpy(3,-2.0,a,2,b,2)
--R
--R
--R (17) [- 1.,2.,- 3.,4.,- 5.,6.,7.]

```

```

--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 55

--S 56 of 208
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R (18) [1.,2.,3.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 56

--S 57 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R (19) [1.,2.,3.,4.,5.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 57

--S 58 of 208
daxpy(3,-2.0,a,1,b,2)
--R
--R
--R (20) [- 1.,2.,- 1.,4.,- 1.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 58

--S 59 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (21) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 59

--S 60 of 208
daxpy(3,0.0,a,1,b,2)
--R
--R
--R (22) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 60

)clear all

--S 61 of 208
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (1) [1.,2.,3.,4.,5.,6.,7.]

```

```

--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 61

--S 62 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (2) [0.,0.,0.,0.,0.,0.,0.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 62

--S 63 of 208
dcopy(3,a,1,b,1)
--R
--R
--R (3) [1.,2.,3.,0.,0.,0.,0.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 63

--S 64 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (4) [0.,0.,0.,0.,0.,0.,0.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 64

--S 65 of 208
dcopy(7,a,1,b,1)
--R
--R
--R (5) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 65

--S 66 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (6) [0.,0.,0.,0.,0.,0.,0.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 66

--S 67 of 208
dcopy(8,a,1,b,1)
--R
--R
--R (7) [0.,0.,0.,0.,0.,0.,0.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 67

```

```

--S 68 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (8) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 68

--S 69 of 208
dcopy(3,a,3,b,3)
--R
--R
--R (9) [1.,0.,0.,4.,0.,0.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 69

--S 70 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (10) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 70

--S 71 of 208
dcopy(4,a,2,b,2)
--R
--R
--R (11) [1.,0.,3.,0.,5.,0.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 71

--S 72 of 208
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (12) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 72

--S 73 of 208
dcopy(5,a,2,b,2)
--R
--R
--R (13) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 73

--S 74 of 208

```

```

b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R   (14)  [0.,0.,0.,0.,0.,0.,0.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 74

--S 75 of 208
dcopy(3,a,2,b,2)
--R
--R
--R   (15)  [1.,0.,3.,0.,5.,0.,0.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 75

--S 76 of 208
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R   (16)  [1.,2.,3.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 76

--S 77 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (17)  [1.,2.,3.,4.,5.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 77

--S 78 of 208
dcopy(3,a,1,b,1)
--R
--R
--R   (18)  [1.,2.,3.,4.,5.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 78

--S 79 of 208
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (19)  [1.,2.,3.,4.,5.]
--R                                             Type: PrimitiveArray(DoubleFloat)
--E 79

--S 80 of 208
dcopy(3,a,1,b,2)
--R

```


--E 86

--S 87 of 208

ddot(3,a,1,b,1)

--R

--R (4) 38.

--R

Type: DoubleFloat

--E 87

--S 88 of 208

ddot(3,a,1,b,2)

--R

--R (5) 46.

--R

Type: DoubleFloat

--E 88

--S 89 of 208

ddot(3,a,2,b,1)

--R

--R (6) 58.

--R

Type: DoubleFloat

--E 89

--S 90 of 208

ddot(3,a,1,b,-2)

--R

--R (7) 38.

--R

Type: DoubleFloat

--E 90

--S 91 of 208

ddot(3,a,-2,b,1)

--R

--R (8) 50.

--R

Type: DoubleFloat

--E 91

--S 92 of 208

ddot(3,a,-2,b,-2)

--R

--R (9) 71.

--R

Type: DoubleFloat

--E 92

)clear all

--S 93 of 208

a:PRIMARR(DFLOAT):=[[3.0, -4.0, 5.0, -7.0, 9.0]]

--R

--R (1) [3., -4., 5., -7., 9.]

```

--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 93

--S 94 of 208
dnrm2(3,a,1)
--R
--R (2) 7.0710678118654755
--R                                                    Type: DoubleFloat
--E 94

--S 95 of 208
dnrm2(5,a,1)
--R
--R (3) 13.416407864998739
--R                                                    Type: DoubleFloat
--E 95

--S 96 of 208
dnrm2(3,a,2)
--R
--R (4) 10.723805294763608
--R                                                    Type: DoubleFloat
--E 96

)clear all
--S 97 of 208
a:MATRIX(DFLOAT):=[[6,5,0],[5,1,4],[0,4,3]]
--R
--R      +6.  5.  0.+
--R      |      |
--R (1) |5.  1.  4.|
--R      |      |
--R      +0.  4.  3.+
--R                                                    Type: Matrix(DoubleFloat)
--E 97

--S 98 of 208
t1:=drotg(elt(a,1,1),elt(a,1,2),0.0,0.0)
--R
--R (2)
--R [7.810249675906654, 0.64018439966447993, 0.76822127959737585,
--R  0.64018439966447993]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 98

--S 99 of 208
g1:MATRIX(DFLOAT):=[[elt(t1,2), elt(t1,3),0.0],_
                    [-elt(t1,3),elt(t1,2),0.0],_
                    [0.0,      0.0,      1.0]]
--R

```

```

--R
--R      + 0.76822127959737585  0.64018439966447993  0.+
--R      |
--R      (3) |- 0.64018439966447993  0.76822127959737585  0.|
--R      |
--R      + 0. 0. 1.+
--R
--R                                          Type: Matrix(DoubleFloat)
--E 99

```

```

--S 100 of 208
t2:=g1*a
--R
--R      + 7.810249675906654  4.4812907976513596  2.5607375986579197+
--R      |
--R      (4) |- 4.4408920985006262E-16 - 2.4327007187250236  3.0728851183895034|
--R      |
--R      + 0. 4. 3. +
--R
--R                                          Type: Matrix(DoubleFloat)
--E 100

```

```

--S 101 of 208
t3:=drotg(elt(t2,2,2),elt(a,3,2),0.0,0.0)
--R
--R      (5)
--R      [4.6816698716254272, - 1.924474241977076, - 0.51962243930719854,
--R      0.85439599751428896]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 101

```

```

--S 102 of 208
g2:=MATRIX(DFLOAT):=[[1.0, 0.0, 0.0],_
                     [0.0, elt(t3,2),elt(t3,3)],_
                     [0.0,-elt(t3,3),elt(t3,2)]]
--R
--R
--R      +1. 0. 0. +
--R      |
--R      (6) |0. - 0.51962243930719854  0.85439599751428896 |
--R      |
--R      +0. - 0.85439599751428896 - 0.51962243930719854+
--R
--R                                          Type: Matrix(DoubleFloat)
--E 102

```

```

--S 103 of 208
g2*g1*a
--R
--R      + 7.810249675906654  4.4812907976513596  2.5607375986579197 +
--R      |
--R      (7) | 2.2204460492503131E-16  4.6816698716254272  0.96644793161452336 |
--R      |

```

```

--R      +- 4.4408920985006262E-16      0.      - 4.1843280638948093+
--R                                          Type: Matrix(DoubleFloat)
--E 103

--S 104 of 208
q:=transpose(g1)*transpose(g2)
--R
--R      +0.76822127959737585      0.33265417936007158      0.54697098874441952 +
--R      |
--R      (8) |0.64018439966447993 - 0.39918501523208583 - 0.65636518649330344|
--R      |
--R      +      0.      0.85439599751428896 - 0.51962243930719854+
--R                                          Type: Matrix(DoubleFloat)
--E 104

)clear all

--S 105 of 208
dx:PRIMARR(DFLOAT):=[[6,0, 1.0, 4.0, -1.0, -1.0]]
--R
--R      (1) [6.,0.,1.,4.,- 1.,- 1.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 105

--S 106 of 208
dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]
--R
--R      (2) [5.,1.,- 4.,4.,- 4.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 106

--S 107 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
--R
--R      (3)
--R      [
--R      [7.778174591, 0.70710678100000002, - 2.1213203429999998,
--R      5.6568542480000001, - 3.5355339050000003, - 1.]
--R      ,
--R      [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R      - 2.1213203429999998]
--R      ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 107

--S 108 of 208
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R      (4)

```

```

--R  [
--R    [7.778174591, 0.70710678100000002, - 2.1213203429999998,
--R      5.6568542480000001, - 3.5355339050000003, - 1.]
--R    ,
--R    [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R      - 2.1213203429999998]
--R  ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 108

--S 109 of 208
drot(5,dx,1,dy,1,0.707106781,-0.707106781) -- rotate by -45 degrees
--R
--R  (5)
--R  [
--R    [5.9999999968341839, 7.8496237287950521E-17, 0.99999999947236384,
--R      3.9999999978894558, - 0.99999999947236451, - 1.]
--R    ,
--R    [4.9999999973618188, 0.99999999947236384, - 3.9999999978894558,
--R      3.9999999978894554, - 3.9999999978894554]
--R  ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 109

--S 110 of 208
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R  (6)
--R  [
--R    [5.9999999968341839, 7.8496237287950521E-17, 0.99999999947236384,
--R      3.9999999978894558, - 0.99999999947236451, - 1.]
--R    ,
--R    [4.9999999973618188, 0.99999999947236384, - 3.9999999978894558,
--R      3.9999999978894554, - 3.9999999978894554]
--R  ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 110

--S 111 of 208
dx:PRIMARR(DFLOAT):=[[6,0, 1.0, 4.0, -1.0, -1.0]]
--R
--R  (7)  [6.,0.,1.,4.,- 1.,- 1.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 111

--S 112 of 208
dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]

```

```

--R
--R (8) [5.,1.,- 4.,4.,- 4.]
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 112

--S 113 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
--R
--R (9)
--R [
--R [7.778174591, 0.70710678100000002, - 2.1213203429999998,
--R 5.6568542480000001, - 3.5355339050000003, - 1.]
--R ,
--R [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R - 2.1213203429999998]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 113

--S 114 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 90 degrees
--R
--R (10)
--R [
--R [4.9999999973618197, 0.99999999947236395, - 3.9999999978894558,
--R 3.9999999978894558, - 3.9999999978894558, - 1.]
--R ,
--R [- 5.9999999968341839, 0., - 0.99999999947236429, - 3.9999999978894558,
--R 0.99999999947236429]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 114

--S 115 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 135 degrees
--R
--R (11)
--R [
--R [- 0.70710678062690524, 0.70710678062690502, - 3.535533903134525, 0.,
--R - 2.1213203418807147, - 1.]
--R ,
--R [- 7.7781745868959549, - 0.70710678062690502, 2.1213203418807147,
--R - 5.6568542450152401, 3.535533903134525]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 115

```

```

--S 116 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 180 degrees
--R
--R (12)
--R [
--R [- 5.9999999936683679, 0., - 0.99999999894472813, - 3.9999999957789121,
--R 0.99999999894472813, - 1.]
--R ,
--R [- 4.9999999947236393, - 0.99999999894472802, 3.9999999957789116,
--R - 3.9999999957789121, 3.9999999957789116]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 116

```

```

--S 117 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 225 degrees
--R
--R (13)
--R [
--R [- 7.7781745827919098, - 0.70710678025381002, 2.1213203407614296,
--R - 5.6568542420304802, 3.5355339012690501, - 1.]
--R ,
--R [0.70710678025381046, - 0.70710678025381002, 3.5355339012690501, 0.,
--R 2.1213203407614296]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 117

```

```

--S 118 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 270 degrees
--R
--R (14)
--R [
--R [- 4.999999992085459, - 0.99999999841709197, 3.9999999936683674,
--R - 3.9999999936683679, 3.9999999936683674, - 1.]
--R ,
--R [5.9999999905025518, 0., 0.99999999841709231, 3.9999999936683679,
--R - 0.99999999841709231]
--R ]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 118

```

```

--S 119 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 315 degrees
--R
--R (15)
--R [

```

```

--R      [0.70710677988071569, - 0.70710677988071502, 3.5355338994035752, 0.,
--R      2.1213203396421445, - 1.]
--R      ,
--R
--R      [7.7781745786878647, 0.70710677988071502, - 2.1213203396421445,
--R      5.6568542390457202, - 3.5355338994035752]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 119

--S 120 of 208
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 360 degrees
--R
--R      (16)
--R      [
--R      [5.9999999873367358, 0., 0.99999999788945637, 3.9999999915578237,
--R      - 0.99999999788945637, - 1.]
--R      ,
--R      [4.9999999894472786, 0.99999999788945593, - 3.9999999915578233,
--R      3.9999999915578237, - 3.9999999915578233]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 120

--S 121 of 208
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R      (17)
--R      [
--R      [5.9999999873367358, 0., 0.99999999788945637, 3.9999999915578237,
--R      - 0.99999999788945637, - 1.]
--R      ,
--R      [4.9999999894472786, 0.99999999788945593, - 3.9999999915578233,
--R      3.9999999915578237, - 3.9999999915578233]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 121

)clear all
--S 122 of 208
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
--R
--R      (1)  [1.,2.,3.,4.,5.,6.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 122

--S 123 of 208
dscal(6,2.0,dx,1)

```



```

--R
--R  (2)  [2.,4.,6.,8.,10.,12.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 123

--S 124 of 208
dx
--R
--R  (3)  [2.,4.,6.,8.,10.,12.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 124

--S 125 of 208
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
--R
--R  (4)  [1.,2.,3.,4.,5.,6.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 125

--S 126 of 208
dscal(3,0.5,dx,1)
--R
--R  (5)  [0.5,1.,1.5,4.,5.,6.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 126

--S 127 of 208
dx
--R
--R  (6)  [0.5,1.,1.5,4.,5.,6.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 127

)clear all

--S 128 of 208
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R  (1)  [1.,2.,3.,4.,5.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 128

--S 129 of 208
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R  (2)  [9.,8.,7.,6.,- 5.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 129

--S 130 of 208

```

```

dswap(5,dx,1,dy,1)
--R
--R (3) [[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 130

--S 131 of 208
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R (4) [1.,2.,3.,4.,5.]
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 131

--S 132 of 208
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R (5) [9.,8.,7.,6.,- 5.]
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 132

--S 133 of 208
dswap(3,dx,2,dy,2)
--R
--R (6) [[9.,2.,7.,4.,- 5.],[1.,8.,3.,6.,5.]]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 133

--S 134 of 208
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R (7) [1.,2.,3.,4.,5.]
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 134

--S 135 of 208
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R (8) [9.,8.,7.,6.,- 5.]
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 135

--S 136 of 208
dswap(5,dx,1,dy,-1)
--R
--R (9) [[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 136

)clear all

```



```

--E 143

--S 144 of 208
dznrm2(3,a,1) -- should be 11.269
--R
--R (4) 11.269427669584644
--R
--R                                          Type: DoubleFloat
--E 144

--S 145 of 208
dznrm2(3,a,-1) -- should be 0.0
--R
--R (5) 0.
--R
--R                                          Type: DoubleFloat
--E 145

--S 146 of 208
dznrm2(-3,a,-1) -- should be 0.0
--R
--R (6) 0.
--R
--R                                          Type: DoubleFloat
--E 146

--S 147 of 208
dznrm2(1,a,1) -- should be 5.0
--R
--R (7) 5.
--R
--R                                          Type: DoubleFloat
--E 147

--S 148 of 208
dznrm2(1,a,2) -- should be 5.0
--R
--R (8) 5.
--R
--R                                          Type: DoubleFloat
--E 148

)clear all

--S 149 of 208
a:PRIMARR(COMPLEX(FLOAT))
--R
--R                                          Type: Void
--E 149

--S 150 of 208
a:=[[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]
--R
--R (2) [3.0 + 4.0 %i,- 4.0 + 5.0 %i,5.0 + 6.0 %i,7.0 - 8.0 %i,- 9.0 - 2.0 %i]
--R
--R                                          Type: PrimitivesArray(Complex(Float))
--E 150

```

[illegible]

--E 157

--S 158 of 208

idamax(3,a,1) -- should be 1

--R

--R (3) 1

--R

Type: PositiveInteger

--E 158

--S 159 of 208

idamax(0,a,1) -- should be -1

--R

--R (4) - 1

--R

Type: Integer

--E 159

--S 160 of 208

idamax(-5,a,1) -- should be -1

--R

--R (5) - 1

--R

Type: Integer

--E 160

--S 161 of 208

idamax(5,a,-1) -- should be -1

--R

--R (6) - 1

--R

Type: Integer

--E 161

--S 162 of 208

idamax(5,a,2) -- should be 0

--R

--R (7) 0

--R

Type: NonNegativeInteger

--E 162

--S 163 of 208

idamax(1,a,0) -- should be -1

--R

--R (8) - 1

--R

Type: Integer

--E 163

--S 164 of 208

idamax(1,a,-1) -- should be -1

--R

--R (9) - 1

--R

Type: Integer

--E 164

```

--S 165 of 208
a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
--R
--R   (10)  [3.,4.,- 3.,- 5.,- 1.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 165

--S 166 of 208
idamax(5,a,1) -- should be 3
--R
--R   (11)  3
--R
--R                                          Type: PositiveInteger
--E 166

)clear all

--S 167 of 208
a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
--R
--R
--R   (1)  [3.0,4.0,- 3.0,5.0,- 1.0]
--R
--R                                          Type: PrimitiveArray(Float)
--E 167

--S 168 of 208
isamax(5,a,1) -- should be 3
--R
--R   (2)  3
--R
--R                                          Type: PositiveInteger
--E 168

--S 169 of 208
isamax(3,a,1) -- should be 1
--R
--R   (3)  1
--R
--R                                          Type: PositiveInteger
--E 169

--S 170 of 208
isamax(0,a,1) -- should be -1
--R
--R   (4)  - 1
--R
--R                                          Type: Integer
--E 170

--S 171 of 208
isamax(-5,a,1) -- should be -1
--R
--R   (5)  - 1

```

```

--R                                                    Type: Integer
--E 171

--S 172 of 208
isamax(5,a,-1) -- should be -1
--R
--R   (6)  - 1
--R
--R                                                    Type: Integer
--E 172

--S 173 of 208
isamax(5,a,2) -- should be 0
--R
--R   (7)  0
--R
--R                                                    Type: NonNegativeInteger
--E 173

--S 174 of 208
isamax(1,a,0) -- should be -1
--R
--R   (8)  - 1
--R
--R                                                    Type: Integer
--E 174

--S 175 of 208
isamax(1,a,-1) -- should be -1
--R
--R   (9)  - 1
--R
--R                                                    Type: Integer
--E 175

--S 176 of 208
a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
--R
--R
--R   (10)  [3.0,4.0,- 3.0,- 5.0,- 1.0]
--R
--R                                                    Type: PrimitiveArray(Float)
--E 176

--S 177 of 208
isamax(5,a,1) -- should be 3
--R
--R   (11)  3
--R
--R                                                    Type: PositiveInteger
--E 177

)clear all

--S 178 of 208
a:PRIMARR(COMPLEX(DFLOAT)):= [[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]

```



```

--R
--R (1) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrIMITIVEArray(Complex(DoubleFloat))
--E 178

--S 179 of 208
izamax(5,a,1) -- should be 3
--R
--R (2) 3
--R                                         Type: PositiveInteger
--E 179

--S 180 of 208
izamax(0,a,1) -- should be -1
--R
--R (3) - 1
--R                                         Type: Integer
--E 180

--S 181 of 208
izamax(5,a,-1) -- should be -1
--R
--R (4) - 1
--R                                         Type: Integer
--E 181

--S 182 of 208
izamax(3,a,1) -- should be 2
--R
--R (5) 2
--R                                         Type: PositiveInteger
--E 182

--S 183 of 208
izamax(3,a,2) -- should be 1
--R
--R (6) 1
--R                                         Type: PositiveInteger
--E 183

)clear all

--S 184 of 208
a:PRIMARR(COMPLEX(DFLOAT)):=_
[[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (1) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrIMITIVEArray(Complex(DoubleFloat))
--E 184

```

```

--S 185 of 208
b:PRIMARR(COMPLEX(DFLOAT)):=_
  [[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (2) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 185

--S 186 of 208
zaxpy(3,2.0,a,1,b,1)
--R
--R (3) [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 186

--S 187 of 208
b:=[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (4) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 187

--S 188 of 208
zaxpy(5,2.0,a,1,b,1)
--R
--R (5) [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,21. - 24. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 188

--S 189 of 208
b:=[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (6) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 189

--S 190 of 208
zaxpy(3,2.0,a,3,b,3)
--R
--R (7) [9. + 12. %i,- 4. + 5. %i,5. + 6. %i,21. - 24. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 190

--S 191 of 208
b:=[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (8) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 191

```

```

--S 192 of 208
zaxpy(4,2.0,a,2,b,2)
--R
--R (9) [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 192

--S 193 of 208
b:=[3.0+4.0%i, -4.0+5.0%i, 5.0+6.0%i, 7.0-8.0%i, -9.0-2.0%i]
--R
--R (10) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 193

--S 194 of 208
zaxpy(3,2.0,a,2,b,2)
--R
--R (11) [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 194

--S 195 of 208
b:=[3.0+4.0%i, -4.0+5.0%i, 5.0+6.0%i, 7.0-8.0%i, -9.0-2.0%i]
--R
--R (12) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 195

--S 196 of 208
zaxpy(3,-2.0,a,1,b,2)
--R
--R (13) [- 3. - 4. %i,- 4. + 5. %i,13. - 4. %i,7. - 8. %i,- 19. - 14. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 196

--S 197 of 208
b:=[3.0+4.0%i, -4.0+5.0%i, 5.0+6.0%i, 7.0-8.0%i, -9.0-2.0%i]
--R
--R (14) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 197

--S 198 of 208
zaxpy(3,2.0,a,1,b,2)
--R
--R (15) [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 198

--S 199 of 208

```

```

b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (16) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 199

--S 200 of 208
zaxpy(-3,2.0,a,1,b,2)
--R
--R (17) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 200

--S 201 of 208
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (18) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 201

--S 202 of 208
zaxpy(3,2.0,a,-1,b,2)
--R
--R (19) [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 202

--S 203 of 208
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (20) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 203

--S 204 of 208
zaxpy(3,2.0,a,1,b,-2)
--R
--R (21) [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 204

--S 205 of 208
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (22) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 205

--S 206 of 208
zaxpy(3,2.0,a,-1,b,-2)

```

```

--R
--R (23) [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 206

--S 207 of 208
b:=[[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (24) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 207

--S 208 of 208
zaxpy(3,0.0,a,1,b,1)
--R
--R (25) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 208

)spool
)lisp (bye)

```

— BlasLevelOne.help —

=====

BlasLevelOne examples

=====

The dcabs1 routine computes the sum of the absolute value of the real and imaginary parts of a complex number.

```

t1:Complex DoubleFloat := complex(1.0,0)
1.

dcabs1(t1)
1.

t2:Complex DoubleFloat := complex(1.0,1.0)
1. + %i

dcabs1(t2)
2.

t3:Complex DoubleFloat := complex(1.0,-1.0)
1. - %i

dcabs1(t3)

```

```

2.

t4:Complex DoubleFloat := complex(-1.0,-1.0)
    - 1. - %i

dcabs1(t4)
    2.

t5:Complex DoubleFloat := complex(-2.0,-2.0)
    - 2. - 2. %i

dcabs1(t5)
    4.

See Also:
o )show BlasLevelOne

```

BlasLevelOne (BLAS1)

— package BLAS1 BlasLevelOne —

```

)abbrev package BLAS1 BlasLevelOne
++ Author: Timothy Daly
++ Date Created: 2010
++ Date March 24, 2010
++ Description:
++ This package provides an interface to the Blas library (level 1)
BlasLevelOne() : Exports == Implementation where

SI    ==> SingleInteger
INT   ==> Integer
DF    ==> DoubleFloat
SX    ==> PrimitiveArray(Float)
DX    ==> PrimitiveArray(DoubleFloat)
CDF   ==> Complex(DoubleFloat)
LDX   ==> List(PrimitiveArray(DoubleFloat))
PCDF  ==> PrimitiveArray(Complex(DoubleFloat))
PCF   ==> PrimitiveArray(Complex(Float))

Exports == with

dcabs1: CDF -> DF
    ++dcabs1(z) computes (+ (abs (realpart z)) (abs (imagpart z)))

```

```

++
++X t1:Complex DoubleFloat := complex(1.0,0)
++X dcabs1(t1)

dasum: (SI, DX, SI) -> DF
++dasum(n,array,incx) computes the sum of n elements in array
++using a stride of incx
++
++X dx:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0,6.0]]
++X dasum(6,dx,1)
++X dasum(3,dx,2)

daxpy: (SI, DF, DX, SI, DX, SI) -> DX
++daxpy(n,da,x,incx,y,incy) computes a y = a*x + y
++for each of the chosen elements of the vectors x and y
++and a constant multiplier a
++Note that the vector y is modified with the results.
++
++X x:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0,6.0]]
++X y:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0,6.0]]
++X daxpy(6,2.0,x,1,y,1)
++X y
++X m:PRIMARR(DFLOAT):=[[1.0,2.0,3.0]]
++X n:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0,6.0]]
++X daxpy(3,-2.0,m,1,n,2)
++X n

dcopy: (SI, DX, SI, DX, SI) -> DX
++dcopy(n,x,incx,y,incy) copies y from x
++for each of the chosen elements of the vectors x and y
++Note that the vector y is modified with the results.
++
++X x:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0,6.0]]
++X y:PRIMARR(DFLOAT):=[[0.0,0.0,0.0,0.0,0.0,0.0]]
++X dcopy(6,x,1,y,1)
++X y
++X m:PRIMARR(DFLOAT):=[[1.0,2.0,3.0]]
++X n:PRIMARR(DFLOAT):=[[0.0,0.0,0.0,0.0,0.0,0.0]]
++X dcopy(3,m,1,n,2)
++X n

ddot: (SI, DX, SI, DX, SI) -> DF
++ddot(n,x,incx,y,incy) computes the vector dot product
++of elements from the vector x and the vector y
++If the indicies are negative the elements are taken
++relative to the far end of the vector.
++
++X x:PRIMARR(DFLOAT):=[[1.0,2.0,3.0,4.0,5.0]]
++X y:PRIMARR(DFLOAT):=[[5.0,6.0,7.0,8.0,9.0]]
++X ddot(0,a,1,b,1) -- handle 0 elements ==> 0

```

```

++X ddot(3,a,1,b,1) -- (1,2,3) * (5,6,7) ==> 38.0
++X ddot(3,a,1,b,2) -- increment = 2 in b (1,2,3) * (5,7,9) ==> 46.0
++X ddot(3,a,2,b,1) -- increment = 2 in a (1,3,5) * (5,6,7) ==> 58.0
++X ddot(3,a,1,b,-2) -- increment = -2 in b (1,2,3) * (9,7,5) ==> 38.0
++X ddot(2,a,-2,b,1) -- increment = -2 in a (5,3,1) * (5,6,7) ==> 50.0
++X ddot(3,a,-2,b,-2) -- (5,3,1) * (9,7,5) ==> 71.0

dnrm2: (SI, DX, SI) -> DF
++dnrm2 takes the norm of the vector, ||x||
++
++X a:PRIMARR(DFLOAT):=[[3.0, -4.0, 5.0, -7.0, 9.0]]
++X dnrm2(3,a,1) -- 7.0710678118654755 = sqrt(3.0^2 + -4.0^2 + 5.0^2)
++X dnrm2(5,a,1) -- 13.416407864998739 = sqrt(180.0)
++X dnrm2(3,a,2) -- 10.72380529476361 = sqrt(115.0)

drotg: (DF, DF, DF, DF) -> DX
++drotg computes a 2D plane Givens rotation spanned by two
++coordinate axes.
++
++X a:MATRIX(DFLOAT):=[[6,5,0],[5,1,4],[0,4,3]]
++X drotg(elt(a,1,1),elt(a,1,2),0.0D0,0.0D0)

drot: (SI, DX, SI, DX, SI, DF, DF) -> LDX
++drot computes a 2D plane Givens rotation spanned by two
++coordinate axes. It modifies the arrays in place.
++The call drot(n,dx,incx,dy,incy,c,s) has the dx array which
++contains the y axis locations and dy which contains the
++y axis locations. They are rotated in parallel where
++c is the cosine of the angle and s is the sine of the angle and
++c^2+s^2 = 1
++
++X dx:PRIMARR(DFLOAT):=[[6.0, 1.0, 4.0, -1.0, -1.0]]
++X dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]
++X drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
++X dx -- dx has been modified
++X dy -- dy has been modified
++X drot(5,dx,1,dy,1,0.707106781,-0.707106781) -- rotate by -45 degrees
++X dx -- dx has been modified
++X dy -- dy has been modified

dscal: (SI, DF, DX, SI) -> DX
++dscal scales each element of the vector by the scalar so
++dscal(n,da,dx,incx) = da*dx for n elements, incremented by incx
++Note that the dx array is modified in place.
++
++X dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
++X dscal(6,2.0,dx,1)
++X dx
++X dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
++X dscal(3,0.5,dx,1)

```



```
++X dx
```

```
dswap: (SI, DX, SI, DX, SI) -> LDX
```

```
++dswap swaps elements from the first vector with the second
```

```
++Note that the arrays are modified in place.
```

```
++
```

```
++X dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dy:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dswap(5,dx,1,dy,1)
```

```
++X dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dy:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dswap(3,dx,2,dy,2)
```

```
++X dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dy:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
++X dswap(5,dx,1,dy,-1)
```

```
dzasum: (SI, PCDF, SI) -> DF
```

```
++dzasum takes the sum over all of the array where each
```

```
++element of the array sum is the sum of the absolute
```

```
++value of the real part and the absolute value of the
```

```
++imaginary part of each array element:
```

```
++ for i in array do sum = sum + (real(a(i)) + imag(a(i)))
```

```
++
```

```
++X d:PRIMARR(COMPLEX(DFLOAT)):[[1.0+2.0*i,-3.0+4.0*i,5.0-6.0*i]]
```

```
++X dzasum(3,d,1) -- 21.0
```

```
++X dzasum(3,d,2) -- 14.0
```

```
++X dzasum(-3,d,1) -- 0.0
```

```
dznrm2: (SI, PCDF, SI) -> DF
```

```
++dznrm2 returns the norm of a complex vector. It computes
```

```
++sqrt(sum(v*conjugate(v)))
```

```
++
```

```
++X a:PRIMARR(COMPLEX(DFLOAT))
```

```
++X a:[[3.+4.*i,-4.+5.*i,5.+6.*i,7.-8.*i,-9.-2.*i]]
```

```
++X dznrm2(5,a,1) -- should be 18.028
```

```
++X dznrm2(3,a,2) -- should be 13.077
```

```
++X dznrm2(3,a,1) -- should be 11.269
```

```
++X dznrm2(3,a,-1) -- should be 0.0
```

```
++X dznrm2(-3,a,-1) -- should be 0.0
```

```
++X dznrm2(1,a,1) -- should be 5.0
```

```
++X dznrm2(1,a,2) -- should be 5.0
```

```
icamax: (INT, PCF, INT) -> INT
```

```
++icamax computes the largest absolute value of the elements
```

```
++of the array and returns the index of the first instance
```

```
++of the maximum
```

```
++
```

```
++X a:PRIMARR(COMPLEX(FLOAT))
```

```
++X a:[[3.+4.*i,-4.+5.*i,5.+6.*i,7.-8.*i,-9.-2.*i]]
```

```
++X icamax(5,a,1) -- should be 3
```

```

++X icamax(0,a,1) -- should be -1
++X icamax(5,a,-1) -- should be -1
++X icamax(3,a,1) -- should be 2
++X icamax(3,a,2) -- should be 1

idamax: (INT, DX, INT) -> INT
++idamax computes the largest absolute value of the elements
++of the array and returns the index of the first instance
++of the maximum.
++
++X a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
++X idamax(5,a,1) -- should be 3
++X idamax(3,a,1) -- should be 1
++X idamax(0,a,1) -- should be -1
++X idamax(-5,a,1) -- should be -1
++X idamax(5,a,-1) -- should be -1
++X idamax(5,a,2) -- should be 0
++X idamax(1,a,0) -- should be -1
++X idamax(1,a,-1) -- should be -1
++X a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
++X idamax(5,a,1) -- should be 3

isamax: (INT, SX, INT) -> INT
++isamax computes the largest absolute value of the elements
++of the array and returns the index of the first instance
++of the maximum.
++
++X a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
++X isamax(5,a,1) -- should be 3
++X isamax(3,a,1) -- should be 1
++X isamax(0,a,1) -- should be -1
++X isamax(-5,a,1) -- should be -1
++X isamax(5,a,-1) -- should be -1
++X isamax(5,a,2) -- should be 0
++X isamax(1,a,0) -- should be -1
++X isamax(1,a,-1) -- should be -1
++X a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
++X isamax(5,a,1) -- should be 3

izamax: (SI, PCDF, SI) -> INT
++izamax computes the largest absolute value of the elements
++of the array and returns the index of the first instance
++of the maximum.
++
++X a:PRIMARR(COMPLEX(DFLOAT))
++X a:=[[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]
++X izamax(5,a,1) -- should be 3
++X izamax(0,a,1) -- should be -1
++X izamax(5,a,-1) -- should be -1
++X izamax(3,a,1) -- should be 2

```

```
++X izamax(3,a,2)  -- should be 1
```

```
zaxpy: (SI, CDF, PCDF, SI, PCDF, SI) -> PCDF
++zaxpy(n,da,x,incx,y,incy) computes a y = a*x + y
++for each of the chosen elements of the vectors x and y
++and a constant multiplier a
++Note that the vector y is modified with the results.
++
++X a:PRIMARR(COMPLEX(DFLOAT))
++X a:=[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]
++X b:PRIMARR(COMPLEX(DFLOAT))
++X b:=[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]
++X zaxpy(3,2.0,a,1,b,1)
++X b:=[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]
++X zaxpy(5,2.0,a,1,b,1)
++X b:=[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]
++X zaxpy(3,2.0,a,3,b,3)
++X b:=[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]
++X zaxpy(4,2.0,a,2,b,2)
```

```
Implementation == add
```

```
dcabs1(z:CDF):DF ==
  DCABS1(COMPLEX(real(z),imag(z))$Lisp)$Lisp
dasum(n:SI,dx:DX,incx:SI):DF ==
  DASUM(n,dx,incx)$Lisp
daxpy(n:SI,da:DF,dx:DX,incx:SI,dy:DX,incy:SI):DX ==
  DAXPY(n,da,dx,incx,dy,incy)$Lisp
dcopy(n:SI,dx:DX,incx:SI,dy:DX,incy:SI):DX ==
  DCOPY(n,dx,incx,dy,incy)$Lisp
ddot(n:SI,dx:DX,incx:SI,dy:DX,incy:SI):DF ==
  DDOT(n,dx,incx,dy,incy)$Lisp
dnrm2(n:SI,dx:DX,incx:SI):DF ==
  DNRM2(n,dx,incx)$Lisp
drotg(a:DF,b:DF,c:DF,s:DF):DX ==
  DROTG(a,b,c,s)$Lisp
drot(n:SI,dx:DX,incx:SI,dy:DX,incy:SI,c:DF,s:DF):LDX ==
  DROT(n,dx,incx,dy,incy,c,s)$Lisp
dscal(n:SI,da:DF,dx:DX,incx:SI):DX ==
  DSCAL(n,da,dx,incx)$Lisp
dswap(n:SI,dx:DX,incx:SI,dy:DX,incy:SI):LDX ==
  DSWAP(n,dx,incx,dy,incy)$Lisp
dzasum(n:SI,dz:PCDF,incx:SI):DF ==
  DZASUMSPAD(n,dz,incx)$Lisp
dznrm2(n:SI,dz:PCDF,incx:SI):DF ==
  DZNRM2SPAD(n,dz,incx)$Lisp
icamax(n:INT,dz:PCF,incx:INT):INT ==
  ICAMAXSPAD(n,dz,incx)$Lisp
idamax(n:INT,dz:DX,incx:INT):INT ==
  IDAMAX(n,dz,incx)$Lisp
```

```

isamax(n:INT,dz:SX,incx:INT):INT ==
  ISAMAXSPAD(n,dz,incx)$Lisp
izamax(n:SI,dz:PCDF,incx:SI):INT ==
  IZAMAXSPAD(n,dz,incx)$Lisp
zaxpy(n:SI,da:CDF,dx:PCDF,incx:SI,dy:PCDF,incy:SI):PCDF ==
  ZAXPYSPAD(n,da,dx,incx,dy,incy)$Lisp

```

— BLAS1.dotabb —

```

"BLAS1" [color="#444488",href="bookvol10.5.pdf#nameddest=BLAS1"]
"FIELD" [color="#4488FF",href="bookvol10.2.pdf#nameddest=FIELD"]
"RADCAT" [color="#4488FF",href="bookvol10.2.pdf#nameddest=RADCAT"]
"BLAS1" -> "FIELD"
"BLAS1" -> "RADCAT"

```

dcabs1 BLAS

— dcabs1.input —

```

)set break resume
)sys rm -f dcabs1.output
)spool dcabs1.output
)set message test on
)set message auto off
)clear all

--S 1 of 10
t1:Complex DoubleFloat := complex(1.0,0)
--R
--R
--R (1) 1.
--R
--R                                          Type: Complex(DoubleFloat)
--E 1

--S 2 of 10
dcabs1(t1)
--R
--R
--R (2) 1.

```



```

--S 9 of 10
t5:Complex DoubleFloat := complex(-2.0,-2.0)
--R
--R
--R (9) - 2. - 2. %i
--R
--R                                         Type: Complex(DoubleFloat)
--E 9

```

```

--S 10 of 10
dcabs1(t5)
--R
--R
--R (10) 4.
--R
--R                                         Type: DoubleFloat
--E 10

```

```

)spool
)lisp (bye)

```

— dcabs1.help —

```

=====
dcabs1 examples
=====

```

The dcabs1 routine computes the sum of the absolute value of the real and imaginary parts of a complex number.

```

t1:Complex DoubleFloat := complex(1.0,0)
1.

dcabs1(t1)
1.

t2:Complex DoubleFloat := complex(1.0,1.0)
1. + %i

dcabs1(t2)
2.

t3:Complex DoubleFloat := complex(1.0,-1.0)
1. - %i

dcabs1(t3)
2.

```

```
t4:Complex DoubleFloat := complex(-1.0,-1.0)
  - 1. - %i
```

```
dcabs1(t4)
  2.
```

```
t5:Complex DoubleFloat := complex(-2.0,-2.0)
  - 2. - 2. %i
```

```
dcabs1(t5)
  4.
```

```
=====
Man Page Details
=====
```

The argument is:

```
\begin{itemize}
\item z - Complex DoubleFloat
\end{itemize}
```

The result is

```
\begin{itemize}
\item (+ (abs (realpart z)) (abs (imagpart z)))
\end{itemize}
```

See Also:

```
o )show BlasLevelOne
o )display operations dcabs1
o )help dcabs1
```

Axiom represents the type Complex(DoubleFloat) as a pair whose car is the real part and whose cdr is the imaginary part. This fact is used in this implementation.

This should really be a macro.

— **dcabs1.f** —

```
double precision function dcabs1(z)
C ORIGINAL:
c      double complex z,zz
c      double precision t(2)
c      equivalence (zz,t(1))
c      zz = z
c      dcabs1 = dabs(t(1)) + dabs(t(2))
c NEW
double complex z
dcabs1 = dabs(dble(z)) + dabs(dimag(z))
```

```

        return
      end

\end{verbatim}

\begin{chunk}{dcabs1 example}
  program dcabs1EX
*    Tim Daly April 23, 2012
*    unit tests for BLAS dcabs1
    double complex a,b,c,d
    a=COMPLEX(2.1,2.1)
    b=(3.1D2,4.1D3)
    c=a+b
    d=dcabs1(c)
    write(6,100)a
100  format("          a=(",f10.3,",",f10.3,")")
    write(6,200)b
200  format("          b=(",f10.3,",",f10.3,")")
    write(6,300)c
300  format("          a+b=(",f10.3,",",f10.3,")")
    write(6,400)d
400  format("dcabs1(c)=(",f10.3,",",f10.3,")")
    stop
  end

```

```
gcc -o dcabs1EX dcabs1EX.f -lgfortran dcabs1.o && ./dcabs1EX
```

```

          a=(      2.100,      2.100)
          b=(    310.000,   4100.000)
        a+b=(    312.100,   4102.100)
dcabs1(c)=(   4414.200,      0.000)

```

— BLAS 1 dcabs1 —

```

(declaim (ftype (function (cons) double-float) dcabs1))
(defun dcabs1 (z)
  ; Tim Daly April 23, 2012
  "Complex(DoubleFloat) z is a pair where (realpart . imaginarypart)
  so the spad caller needs to construct a lisp complex number.
  The result is a DoubleFloat (+ (abs (realpart z)) (abs (imagpart z)))"
  (the double-float
    (+ (the double-float (abs (the double-float (realpart z))))
       (the double-float (abs (the double-float (imagpart z)))))))

```

— BLAS 1 dcabs1 test —

```
(dcabs1 #C(312.100 4102.100))
```

lsame BLAS

The `lsame` function returns `t` if `ca` and `cb` represent the same letter regardless of case.

This has been replaced everywhere with common lisp's `char-equal` function which compares characters ignoring case. The type (simple-array character (*)) has been replaced everywhere with `character`.

— lsame.f —

```

      LOGICAL          FUNCTION LSAME( CA, CB )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    September 30, 1994
*
*    .. Scalar Arguments ..
*      CHARACTER          CA, CB
*    ..
*
*  Purpose
*  =====
*
*  LSAME returns .TRUE. if CA is the same letter as CB regardless of
*  case.
*
*  Arguments
*  =====
*
*  CA      (input) CHARACTER*1
*  CB      (input) CHARACTER*1
*           CA and CB specify the single characters to be compared.
*
*  =====
*
*    .. Intrinsic Functions ..
*      INTRINSIC          ICHAR
*    ..

```

```

*      .. Local Scalars ..
      INTEGER          INTA, INTB, ZCODE
*      ..
*      .. Executable Statements ..
*
*      Test if the characters are equal
*
      LSAME = CA.EQ.CB
      IF( LSAME )
$      RETURN
*
*      Now test for equivalence if both characters are alphabetic.
*
      ZCODE = ICHAR( 'Z' )
*
*      Use 'Z' rather than 'A' so that ASCII can be detected on Prime
*      machines, on which ICHAR returns a value with bit 8 set.
*      ICHAR('A') on Prime machines returns 193 which is the same as
*      ICHAR('A') on an EBCDIC machine.
*
      INTA = ICHAR( CA )
      INTB = ICHAR( CB )
*
      IF( ZCODE.EQ.90 .OR. ZCODE.EQ.122 ) THEN
*
*          ASCII is assumed - ZCODE is the ASCII code of either lower or
*          upper case 'Z'.
*
          IF( INTA.GE.97 .AND. INTA.LE.122 ) INTA = INTA - 32
          IF( INTB.GE.97 .AND. INTB.LE.122 ) INTB = INTB - 32
*
      ELSE IF( ZCODE.EQ.233 .OR. ZCODE.EQ.169 ) THEN
*
*          EBCDIC is assumed - ZCODE is the EBCDIC code of either lower or
*          upper case 'Z'.
*
          IF( INTA.GE.129 .AND. INTA.LE.137 .OR.
$          INTA.GE.145 .AND. INTA.LE.153 .OR.
$          INTA.GE.162 .AND. INTA.LE.169 ) INTA = INTA + 64
          IF( INTB.GE.129 .AND. INTB.LE.137 .OR.
$          INTB.GE.145 .AND. INTB.LE.153 .OR.
$          INTB.GE.162 .AND. INTB.LE.169 ) INTB = INTB + 64
*
      ELSE IF( ZCODE.EQ.218 .OR. ZCODE.EQ.250 ) THEN
*
*          ASCII is assumed, on Prime machines - ZCODE is the ASCII code
*          plus 128 of either lower or upper case 'Z'.
*
          IF( INTA.GE.225 .AND. INTA.LE.250 ) INTA = INTA - 32
          IF( INTB.GE.225 .AND. INTB.LE.250 ) INTB = INTB - 32

```

```

        END IF
        LSAME = INTA.EQ.INTB
*
*   RETURN
*
*   End of LSAME
*
        END

```

\end{verbatim}

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\section{xerbla BLAS}
%\pagehead{xerbla}{xerbla}
%\pagepic{ps/v104algebraicfunction.ps}{AF}{1.00}

```

The {\tt xerbla} routine is an error handler.
It is called if an input parameter has an invalid value.

This function has been rewritten everywhere to use the common lisp error function.

```

\begin{chunk}{xerbla.f}
        SUBROUTINE XERBLA( SRNAME, INFO )
*
*   -- LAPACK auxiliary routine (preliminary version) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   February 29, 1992
*
*   .. Scalar Arguments ..
*   CHARACTER*6          SRNAME
*   INTEGER              INFO
*   ..
*
*   Purpose
*   =====
*
*   XERBLA  is an error handler for the LAPACK routines.
*   It is called by an LAPACK routine if an input parameter has an
*   invalid value.  A message is printed and execution stops.
*
*   Installers may consider modifying the STOP statement in order to
*   call system-specific exception-handling facilities.
*
*   Arguments
*   =====
*
*   SRNAME   (input) CHARACTER*6

```

```

*          The name of the routine which called XERBLA.
*
*  INFO      (input) INTEGER
*             The position of the invalid parameter in the parameter list
*             of the calling routine.
*
*
*          WRITE( *, FMT = 9999 )SRNAME, INFO
*
*
*  * commented out by RLT
*          STOP
*
*  9999 FORMAT( ' ** On entry to ', A6, ' parameter number ', I2, ' had ',
*             $      'an illegal value' )
*
*          End of XERBLA
*
*          END

```

\end{verbatim}

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\chapter{BLAS Level 1}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\section{dasum BLAS}
%\pagehead{dasum}{dasum}
%\pagepic{ps/v104algebraicfunction.ps}{AF}{1.00}

\begin{chunk}{dasum.input}
)set break resume
)sys rm -f dasum.output
)spool dasum.output
)set message test on
)set message auto off
)clear all

--S 1 of 28
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0] ]
--R
--R
--R (1) [1.,2.,3.,4.,5.,6.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 28
dasum(3,a,-1) -- 0.0    neg incx
--R
--R
--R (2) 0.

```



```

--S 9 of 28
dasum(4,a,1) -- 10.0  1.0+2.0+3.0+4.0
--R
--R
--R (9)  10.
--R
--R                                          Type: DoubleFloat
--E 9

--S 10 of 28
dasum(5,a,1) -- 15.0  1.0+2.0+3.0+4.0+5.0
--R
--R
--R (10)  15.
--R
--R                                          Type: DoubleFloat
--E 10

--S 11 of 28
dasum(6,a,1) -- 21.0  1.0+2.0+3.0+4.0+5.0+6.0
--R
--R
--R (11)  21.
--R
--R                                          Type: DoubleFloat
--E 11

--S 12 of 28
dasum(7,a,1) -- 21.0  1.0+2.0+3.0+4.0+5.0+6.0
--R
--R
--R (12)  21.
--R
--R                                          Type: DoubleFloat
--E 12

--S 13 of 28
dasum(1,a,2) -- 1.0  1.0
--R
--R
--R (13)  1.
--R
--R                                          Type: DoubleFloat
--E 13

--S 14 of 28
dasum(2,a,2) -- 4.0  1.0+3.0
--R
--R
--R (14)  4.
--R
--R                                          Type: DoubleFloat
--E 14

--S 15 of 28

```

```
dasum(3,a,2) -- 9.0 1.0+3.0+5.0
```

```
--R
```

```
--R
```

```
--R (15) 9.
```

```
--R
```

Type: DoubleFloat

```
--E 15
```

```
--S 16 of 28
```

```
dasum(4,a,2) -- 9.0 1.0+3.0+5.0
```

```
--R
```

```
--R
```

```
--R (16) 9.
```

```
--R
```

Type: DoubleFloat

```
--E 16
```

```
--S 17 of 28
```

```
dasum(1,a,3) -- 1.0 1.0
```

```
--R
```

```
--R
```

```
--R (17) 1.
```

```
--R
```

Type: DoubleFloat

```
--E 17
```

```
--S 18 of 28
```

```
dasum(2,a,3) -- 5.0 1.0+4.0
```

```
--R
```

```
--R
```

```
--R (18) 5.
```

```
--R
```

Type: DoubleFloat

```
--E 18
```

```
--S 19 of 28
```

```
dasum(3,a,3) -- 5.0 1.0+4.0
```

```
--R
```

```
--R
```

```
--R (19) 5.
```

```
--R
```

Type: DoubleFloat

```
--E 19
```

```
--S 20 of 28
```

```
dasum(1,a,4) -- 1.0 1.0
```

```
--R
```

```
--R
```

```
--R (20) 1.
```

```
--R
```

Type: DoubleFloat

```
--E 20
```

```
--S 21 of 28
```

```
dasum(2,a,4) -- 6.0 1.0+5.0
```

```
--R
```

```

--R
--R (21) 6.
--R
--R                                         Type: DoubleFloat
--E 21

--S 22 of 28
dasum(3,a,4) -- 6.0 1.0+5.0
--R
--R
--R (22) 6.
--R
--R                                         Type: DoubleFloat
--E 22

--S 23 of 28
dasum(1,a,5) -- 1.0 1.0
--R
--R
--R (23) 1.
--R
--R                                         Type: DoubleFloat
--E 23

--S 24 of 28
dasum(2,a,5) -- 7.0 1.0+6.0
--R
--R
--R (24) 7.
--R
--R                                         Type: DoubleFloat
--E 24

--S 25 of 28
dasum(3,a,5) -- 7.0 1.0+6.0
--R
--R
--R (25) 7.
--R
--R                                         Type: DoubleFloat
--E 25

--S 26 of 28
dasum(1,a,6) -- 1.0 1.0
--R
--R
--R (26) 1.
--R
--R                                         Type: DoubleFloat
--E 26

--S 27 of 28
dasum(2,a,6) -- 1.0 1.0
--R
--R
--R (27) 1.

```



```

--R                                                    Type: DoubleFloat
--E 27

--S 28 of 28
dasum(1,a,7) -- 1.0 1.0
--R
--R
--R (28) 1.
--R                                                    Type: DoubleFloat
--E 28

)spool
)lisp (bye)

```

—————

— dasum.help —

```

=====
dasum examples
=====

a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0] ]
[1.,2.,3.,4.,5.,6.]

dasum(3,a,-1) -- 0.0 neg incx
0.

dasum(3,a,0) -- 0.0 zero incx
0.

dasum(-1,a,1) -- 0.0 neg elements
0.

dasum(0,a,1) -- 0.0 no elements
0.

dasum(1,a,1) -- 1.0 1.0
1.

dasum(2,a,1) -- 3.0 1.0+2.0
3.

dasum(3,a,1) -- 6.0 1.0+2.0+3.0
6.

dasum(4,a,1) -- 10.0 1.0+2.0+3.0+4.0
10.

```

```

dasum(5,a,1) -- 15.0  1.0+2.0+3.0+4.0+5.0
15.

dasum(6,a,1) -- 21.0  1.0+2.0+3.0+4.0+5.0+6.0
21.

dasum(7,a,1) -- 21.0  1.0+2.0+3.0+4.0+5.0+6.0
21.

dasum(1,a,2) -- 1.0  1.0
1.

dasum(2,a,2) -- 4.0  1.0+3.0
4.

dasum(3,a,2) -- 9.0  1.0+3.0+5.0
9.

dasum(4,a,2) -- 9.0  1.0+3.0+5.0
9.

dasum(1,a,3) -- 1.0  1.0
1.

dasum(2,a,3) -- 5.0  1.0+4.0
5.

dasum(3,a,3) -- 5.0  1.0+4.0
5.

dasum(1,a,4) -- 1.0  1.0
1.

dasum(2,a,4) -- 6.0  1.0+5.0
6.

dasum(3,a,4) -- 6.0  1.0+5.0
6.

dasum(1,a,5) -- 1.0  1.0
1.

dasum(2,a,5) -- 7.0  1.0+6.0
7.

dasum(3,a,5) -- 7.0  1.0+6.0
7.

dasum(1,a,6) -- 1.0  1.0
1.

```

```
dasum(2,a,6) -- 1.0 1.0
1.
```

```
dasum(1,a,7) -- 1.0 1.0
1.
```

```
=====
Man Page Details
=====
```

Computes doublefloat \$asum $\leftarrow ||\text{re}(x)||_1 + ||\text{im}(x)||_1$

Arguments are:

```
\begin{itemize}
\item n - fixnum
\item dx - array doublefloat
\item incx - fixnum
\end{itemize}
```

Return values are:

```
\begin{itemize}
\item 1 nil
\item 2 nil
\item 3 nil
\end{itemize}
```

NAME

DASUM - BLAS level one, sums the absolute values of the elements of a double precision vector

SYNOPSIS

```
DOUBLE PRECISION FUNCTION DASUM ( n, x, incx )
      INTEGER                n, incx
      DOUBLE PRECISION       x
```

AXIOM SIGNATURE:

```
SI ==> SingleInteger
DF ==> DoubleFloat
DX ==> PrimitiveArray(DoubleFloat)
```

```
dasum: (SI, DX, SI) -> DF
```

DESCRIPTION

This routine performs the following vector operation:

$$\text{DASUM} \leftarrow \sum_{i=1}^n \text{abs}(x(i))$$

ARGUMENTS

n INTEGER. (input)
 Number of vector elements to be summed.
 if $n \leq 0$, the result will be 0.0
 if $n > \text{length}(x)$ then the whole array is summed.

x DOUBLE PRECISION. (input)
 Array of dimension $(n-1) * \text{abs}(\text{incx}) + 1$.
 Vector that contains elements to be summed.

incx INTEGER. (input)
 Increment between elements of x.
 If $\text{incx} \leq 0$, the results will be 0.0

RESULT:

DASUM DOUBLE PRECISION. (output)
 Sum of the absolute values of the elements of the vector x.
 If $n \leq 0$, DASUM is set to 0.0

NOTES:

Axiom uses 0-based arrays. Fortran uses 1-based arrays.

if the index of the array exceeds the length of x
 then no additional elements are added. Thus,
 if $x = \#(1.0 \ 2.0 \ 3.0 \ 4.0 \ 5.0 \ 6.0)$
 then

```
(dasum 3 a -1) = 0.0 ; neg incx
(dasum 3 a 0)  = 0.0 ; zero incx
(dasum -1 a 1) = 0.0 ; neg elements
(dasum 0 a 1)  = 0.0 ; no elements
(dasum 1 a 1)  = 1.0 ; 1.0
(dasum 2 a 1)  = 3.0 ; 1.0+2.0
(dasum 3 a 1)  = 6.0 ; 1.0+2.0+3.0
(dasum 4 a 1)  = 10.0 ; 1.0+2.0+3.0+4.0
(dasum 5 a 1)  = 15.0 ; 1.0+2.0+3.0+4.0+5.0
(dasum 6 a 1)  = 21.0 ; 1.0+2.0+3.0+4.0+5.0+6.0
(dasum 7 a 1)  = 21.0 ; 1.0+2.0+3.0+4.0+5.0+6.0
(dasum 1 a 2)  = 1.0 ; 1.0
(dasum 2 a 2)  = 4.0 ; 1.0+3.0
(dasum 3 a 2)  = 9.0 ; 1.0+3.0+5.0
(dasum 4 a 2)  = 9.0 ; 1.0+3.0+5.0
(dasum 1 a 3)  = 1.0 ; 1.0
(dasum 2 a 3)  = 5.0 ; 1.0+4.0
(dasum 3 a 3)  = 5.0 ; 1.0+4.0
(dasum 1 a 4)  = 1.0 ; 1.0
(dasum 2 a 4)  = 6.0 ; 1.0+5.0
(dasum 3 a 4)  = 6.0 ; 1.0+5.0
(dasum 1 a 5)  = 1.0 ; 1.0
(dasum 2 a 5)  = 7.0 ; 1.0+6.0
(dasum 3 a 5)  = 7.0 ; 1.0+6.0
```

```

      (dasum 1 a 6) = 1.0 ; 1.0
      (dasum 2 a 6) = 1.0 ; 1.0
      (dasum 1 a 7) = 1.0 ; 1.0

```

— dasum.f —

```

      double precision function dasum(n,dx,incx)
c
c      takes the sum of the absolute values.
c      jack dongarra, linpack, 3/11/78.
c      modified 3/93 to return if incx .le. 0.
c      modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dtemp
      integer i,incx,m,mp1,n,nincx
c
      dasum = 0.0d0
      dtemp = 0.0d0
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
         dtemp = dtemp + dabs(dx(i))
10 continue
      dasum = dtemp
      return
c
c      code for increment equal to 1
c
c
c      clean-up loop
c
20 m = mod(n,6)
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
      dtemp = dtemp + dabs(dx(i))
30 continue
   if( n .lt. 6 ) go to 60
40 mp1 = m + 1
   do 50 i = mp1,n,6
      dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
      * + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
50 continue

```

```

60 dasum = dtemp
   return
   end

```

— dasum example —

```

program dasumEX
*   Tim Daly April 24, 2012
*   unit tests for BLAS dasum
   double precision a(6)
   double precision b
   a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)
   write(6,100)a(1),a(2),a(3)
100  format("a(1)=",f6.3," a(2)=",f6.3," a(3)=",f6.3)
   write(6,200)a(4),a(5),a(6)
200  format("a(4)=",f6.3," a(5)=",f6.3," a(6)=",f6.3)
   d=dasum(3,a,-1)
   write(6,300)d
300  format("d=",f6.3," should be 0.0, negative index")
   d=dasum(3,a,0)
   write(6,301)d
301  format("d=",f6.3," should be 0.0, zero increment")
   d=dasum(-11,a,1)
   write(6,302)d
302  format("d=",f6.3," should be 0.0, negative elements")
   d=dasum(0,a,1)
   write(6,303)d
303  format("d=",f6.3," should be 0.0, no elements")
   d=dasum(1,a,1)
   write(6,304)d
304  format("d=",f6.3," should be 1.0")
   d=dasum(2,a,1)
   write(6,305)d
305  format("d=",f6.3," should be 3.0 = 1.0+2.0")
   d=dasum(3,a,1)
   write(6,306)d
306  format("d=",f6.3," should be 6.0 = 1.0+2.0+3.0")
   d=dasum(4,a,1)
   write(6,307)d
307  format("d=",f6.3," should be 10.0 = 1.0+2.0+3.0+4.0")
   d=dasum(5,a,1)
   write(6,308)d
308  format("d=",f6.3," should be 15.0 = 1.0+2.0+3.0+4.0+5.0")
   d=dasum(6,a,1)
   write(6,309)d
309  format("d=",f6.3," should be 21.0 = 1.0+2.0+3.0+4.0+5.0+6.0")

```

```
        d=dasum(7,a,1)
        write(6,310)d
310    format("d=",f6.3," should be 21.0 = 1.0+2.0+3.0+4.0+5.0+6.0")
        d=dasum(1,a,2)
        write(6,311)d
311    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(2,a,2)
        write(6,312)d
312    format("d=",f6.3," should be 4.0 = 1.0+3.0")
        d=dasum(3,a,2)
        write(6,313)d
313    format("d=",f6.3," should be 9.0 = 1.0+3.0+5.0")
        d=dasum(4,a,2)
        write(6,314)d
314    format("d=",f6.3," should be 9.0 = 1.0+3.0+5.0")
        d=dasum(1,a,3)
        write(6,315)d
315    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(2,a,3)
        write(6,316)d
316    format("d=",f6.3," should be 5.0 = 1.0+4.0")
        d=dasum(3,a,3)
        write(6,317)d
317    format("d=",f6.3," should be 5.0 = 1.0+4.0")
        d=dasum(1,a,4)
        write(6,318)d
318    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(2,a,4)
        write(6,319)d
319    format("d=",f6.3," should be 6.0 = 1.0+5.0")
        d=dasum(3,a,4)
        write(6,320)d
320    format("d=",f6.3," should be 6.0 = 1.0+5.0")
        d=dasum(1,a,5)
        write(6,321)d
321    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(2,a,5)
        write(6,322)d
322    format("d=",f6.3," should be 7.0 = 1.0+6.0")
        d=dasum(3,a,5)
        write(6,323)d
323    format("d=",f6.3," should be 7.0 = 1.0+6.0")
        d=dasum(1,a,6)
        write(6,324)d
324    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(2,a,6)
        write(6,325)d
325    format("d=",f6.3," should be 1.0 = 1.0")
        d=dasum(1,a,7)
        write(6,326)d
```

```

326  format("d=",f6.3," should be 1.0 = 1.0")
      stop
      end

```

```

gcc -o dasumEX dasumEX.f -lgfortran dasum.o && ./dasumEX

```

```

a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000
d= 0.000 should be 0.0, negative index
d= 0.000 should be 0.0, zero increment
d= 0.000 should be 0.0, negative elements
d= 0.000 should be 0.0, no elements
d= 1.000 should be 1.0
d= 3.000 should be 3.0 = 1.0+2.0
d= 6.000 should be 6.0 = 1.0+2.0+3.0
d=10.000 should be 10.0 = 1.0+2.0+3.0+4.0
d=15.000 should be 15.0 = 1.0+2.0+3.0+4.0+5.0
d=21.000 should be 21.0 = 1.0+2.0+3.0+4.0+5.0+6.0
d=21.000 should be 21.0 = 1.0+2.0+3.0+4.0+5.0+6.0
d= 1.000 should be 1.0 = 1.0
d= 4.000 should be 4.0 = 1.0+3.0
d= 9.000 should be 9.0 = 1.0+3.0+5.0
d= 9.000 should be 9.0 = 1.0+3.0+5.0
d= 1.000 should be 1.0 = 1.0
d= 5.000 should be 5.0 = 1.0+4.0
d= 5.000 should be 5.0 = 1.0+4.0
d= 1.000 should be 1.0 = 1.0
d= 6.000 should be 6.0 = 1.0+5.0
d= 6.000 should be 6.0 = 1.0+5.0
d= 1.000 should be 1.0 = 1.0
d= 7.000 should be 7.0 = 1.0+6.0
d= 7.000 should be 7.0 = 1.0+6.0
d= 1.000 should be 1.0 = 1.0
d= 1.000 should be 1.0 = 1.0
d= 1.000 should be 1.0 = 1.0

```

— BLAS 1 dasum —

```

(declaim (ftype (function (fixnum (simple-array double-float (*)) fixnum)
                          double-float) dasum))

(defun dasum (n dx incx)
  ; Tim Daly April 24, 2012
  (declare (type (simple-array double-float (*)) dx) (type fixnum incx n))
  (let ((dasum 0.0) (maxlen (length dx)))

```



```
(declare (type (double-float) dasum) (type fixnum maxlen))
(when (> incx 0)
  (when (> n maxlen) (setq n maxlen))
  (unless (<= n 0)
    (do ((i 0 (1+ i)) (j 0 (+ j incx)))
        ((or (>= i n) (>= j maxlen)))
      (setq dasum
        (the double-float
          (+ (the double-float dasum)
             (the double-float (abs (the double-float (svref dx j))))))))))
  dasum))
```

daxpy BLAS

— daxpy.input —

```
)set break resume
)sys rm -f daxpy.output
)spool daxpy.output
)set message test on
)set message auto off
)clear all

--S 1 of 22
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (1) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (2) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 22
daxpy(3,2.0,a,1,b,1)
--R
--R
--R (3) [3.,6.,9.,4.,5.,6.,7.]
```

```

--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 3

--S 4 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (4) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 4

--S 5 of 22
daxpy(7,2.0,a,1,b,1)
--R
--R
--R (5) [3.,6.,9.,12.,15.,18.,21.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 5

--S 6 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (6) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 6

```

Note that Axiom properly handles array indexes that are out of bounds. The BLAS daxpy routine cannot check this condition.

— **daxpy.input** —

```

--S 7 of 22
daxpy(8,2.0,a,1,b,1)
--R
--R
--R (7) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 7

--S 8 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (8) [1.,2.,3.,4.,5.,6.,7.]
--R                                                    Type: PrimitiveArray(DoubleFloat)
--E 8

```

```

--S 9 of 22
daxpy(3,2.0,a,3,b,3)
--R
--R
--R (9) [3.,2.,3.,12.,5.,6.,21.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 9

--S 10 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (10) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 10

--S 11 of 22
daxpy(4,2.0,a,2,b,2)
--R
--R
--R (11) [3.,2.,9.,4.,15.,6.,21.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 11

--S 12 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (12) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 12

--S 13 of 22
daxpy(5,2.0,a,2,b,2)
--R
--R
--R (13) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 13

--S 14 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R (14) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                         Type: PrimitivesArray(DoubleFloat)
--E 14

--S 15 of 22
daxpy(3,2.0,a,2,b,2)

```

```

--R
--R
--R   (15)  [3.,2.,9.,4.,15.,6.,7.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 15

--S 16 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R
--R   (16)  [1.,2.,3.,4.,5.,6.,7.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 16

--S 17 of 22
daxpy(3,-2.0,a,2,b,2)
--R
--R
--R   (17)  [- 1.,2.,- 3.,4.,- 5.,6.,7.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 17

--S 18 of 22
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R
--R   (18)  [1.,2.,3.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 18

--S 19 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (19)  [1.,2.,3.,4.,5.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 19

--S 20 of 22
daxpy(3,-2.0,a,1,b,2)
--R
--R
--R   (20)  [- 1.,2.,- 1.,4.,- 1.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 20

--S 21 of 22
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
--R
--R

```

```

--R (21) [1.,2.,3.,4.,5.,6.,7.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 21

--S 22 of 22
daxpy(3,0.0,a,1,b,2)
--R
--R
--R (22) [1.,2.,3.,4.,5.,6.,7.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 22

)spool
)lisp (bye)

```

— daxpy.help —

=====

daxpy examples

=====

For each of the following examples we assume that we have preset the variables to the following values. Note that the daxpy function will modify the second array. Each example assumes we have reset the variables to these values.

```

a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]

```

then we compute the sum of the first 3 elements of each vector and we show the steps of the computation with trailing comments. The comments show which elements are changed and what the computation is.

```

daxpy(3,2.0,a,1,b,1) ==> [3.,6.,9.,4.,5.,6.,7.]
  dy(0)[3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
  dy(1)[6.0] = dy(1)[2.0] + ( da[2.0] * dx(1)[2.0] )
  dy(2)[9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )

```

or we compute the first 7 elements of each vector

```

daxpy(7,2.0,a,1,b,1) ==> [3.,6.,9.,12.,15.,18.,21.]
  dy(0)[3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
  dy(1)[6.0] = dy(1)[2.0] + ( da[2.0] * dx(1)[2.0] )
  dy(2)[9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
  dy(3)[12.0] = dy(3)[4.0] + ( da[2.0] * dx(3)[4.0] )
  dy(4)[15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )
  dy(5)[18.0] = dy(5)[6.0] + ( da[2.0] * dx(5)[6.0] )

```

```
dy(6)[21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )
```

or we compute the first 8 elements, which fails due to the fact that the vectors are not long enough.

```
daxpy(8,2.0,a,1,b,1) ==> [1.,2.,3.,4.,5.,6.,7.]
```

or we compute the 3 elements, taking every 3rd element, giving the index of 0, 3, 6

```
daxpy(3,2.0,a,3,b,3) ==> [3.,2.,3.,12.,5.,6.,21.]
dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
dy(3) [12.0] = dy(3)[4.0] + ( da[2.0] * dx(3)[4.0] )
dy(6) [21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )
```

or we compute 4 elements, taking every 2nd element, giving the index of 0, 2, 4, 6

```
daxpy(4,2.0,a,2,b,2) ==> [3.,2.,9.,4.,15.,6.,21.]
dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
dy(2) [9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
dy(4) [15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )
dy(6) [21.0] = dy(6)[7.0] + ( da[2.0] * dx(6)[7.0] )
```

or we compute 5 elements, taking every 2nd element, which fails due to vector length

```
daxpy(5,2.0,a,2,b,2) ==> [1.,2.,3.,4.,5.,6.,7.]
```

or we compute 3 elements, taking every 2nd value, giving the index of 0, 2, 4

```
daxpy(3,2.0,a,2,b,2) ==> [3.,2.,9.,4.,15.,6.,7.]
dy(0) [3.0] = dy(0)[1.0] + ( da[2.0] * dx(0)[1.0] )
dy(2) [9.0] = dy(2)[3.0] + ( da[2.0] * dx(2)[3.0] )
dy(4) [15.0] = dy(4)[5.0] + ( da[2.0] * dx(4)[5.0] )
```

or we compute 3 elements, taking every 2nd value, giving the index of 0, 2, 4 but with a negative multiplier

```
daxpy(3,-2.0,a,2,b,2) ==> [- 1.,2.,- 3.,4.,- 5.,6.,7.]
dy(0)[-1.0] = dy(0)[1.0] + ( da[-2.0] * dx(0)[1.0] )
dy(2)[-3.0] = dy(2)[3.0] + ( da[-2.0] * dx(2)[3.0] )
dy(4)[-5.0] = dy(4)[5.0] + ( da[-2.0] * dx(4)[5.0] )
```

or we change the lengths of the input vectors, making them unequal. So for the next two examples we assume the arrays look like:

```
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
```

We compute 3 elements, with a negative multiplier, and different increments using an index for the 'a' array having values 0, 1, 2 using an index for the 'b' array having values 0, 2, 3

```
daxpy(3,-2.0,a,1,b,2) ==> [- 1.,2.,- 1.,4.,- 1.]
  dy(0)[-1.0] = dy(0)[1.0] + ( da[-2.0] * dx(0)[1.0] )
  dy(2)[-1.0] = dy(2)[3.0] + ( da[-2.0] * dx(1)[2.0] )
  dy(4)[-1.0] = dy(4)[5.0] + ( da[-2.0] * dx(2)[3.0] )
```

or we compute 3 elements with a multiplier of 0.0 which fails

```
daxpy(3,0.0,a,1,b,2) ==> [1.,2.,3.,4.,5.,6.,7.]
```

```
=====
Man Page Details
=====
```

NAME

DAXPY - BLAS level one axpy subroutine

SYNOPSIS

```
SUBROUTINE DAXPY      ( n, alpha, x, incx, y, incy )
```

```
      INTEGER          n, incx, incy
```

```
      DOUBLE PRECISION alpha, x, y
```

DESCRIPTION

DAXPY adds a scalar multiple of a double precision vector to another double precision vector.

DAXPY computes a constant alpha times a vector x plus a vector y. The result overwrites the initial values of vector y.

This routine performs the following vector operation:

```
y <-- alpha*x + y
```

incx and incy specify the increment between two consecutive elements of respectively vector x and y.

ARGUMENTS

n INTEGER. (input)
 Number of elements in the vectors. If n <= 0, these routines return without any computation.

alpha DOUBLE PRECISION. (input)
 If alpha = 0 this routine returns without any computation.

x DOUBLE PRECISION, (input)
 Array of dimension (n-1) * |incx| + 1. Contains the vector to
 be scaled before summation.

incx INTEGER. (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

y DOUBLE PRECISION, (input and output)
 array of dimension (n-1) * |incy| + 1.
 Before calling the routine, y contains the vector to be summed.
 After the routine ends, y contains the result of the summation.

incy INTEGER. (input)
 Increment between elements of y.
 If incy = 0, the results will be unpredictable.

RETURN VALUES

When $n \leq 0$, double precision $\alpha = 0.$, this routine returns immediately with no change in its arguments.

— daxpy.f —

```

subroutine daxpy(n,da,dx,incx,dy,incy)
c
c   constant times a vector plus a vector.
c   uses unrolled loops for increments equal to one.
c   jack dongarra, linpack, 3/11/78.
c   modified 12/3/93, array(1) declarations changed to array(*)
c
  double precision dx(*),dy(*),da
  integer i,incx,incy,ix,iy,m,mp1,n
c
  if(n.le.0)return
  if (da .eq. 0.0d0) return
  if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments
c       not equal to 1
c
  ix = 1
  iy = 1
  if(incx.lt.0)ix = (-n+1)*incx + 1
  if(incy.lt.0)iy = (-n+1)*incy + 1
  do 10 i = 1,n

```



```

        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
10 continue
    return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
20 m = mod(n,4)
    if( m .eq. 0 ) go to 40
    do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
30 continue
    if( n .lt. 4 ) return
40 mp1 = m + 1
    do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
50 continue
    return
end

```

Compile with

```

gcc -c daxpy.f
gcc -o daxpyEx daxpyEX.f -lgfortran daxpy.o

```

— daxpy example —

```

program daxpyEX
*   Tim Daly April 24, 2012
*   unit tests for BLAS daxpy (a*x+y)
    double precision a(7)
    double precision b(7)
    double precision c(7)
    a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
    b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
    write(6,100)a(1),a(2),a(3)
100 format("a(1)=",f6.3," a(2)=",f6.3," a(3)=",f6.3)
    write(6,101)a(4),a(5),a(6)
101 format("a(4)=",f6.3," a(5)=",f6.3," a(6)=",f6.3)

```

```

write(6,102)b(1),b(2),b(3)
102 format("b(1)=",f6.3," b(2)=",f6.3," b(3)=",f6.3)
write(6,103)b(4),b(5),b(6)
103 format("b(4)=",f6.3," b(5)=",f6.3," b(6)=",f6.3)

call daxpy(3,2.0d0,a,1,b,1)
write(6,200)
200 format(/,"t200 is (/ 3.0, 6.0, 9.0, 4.0, 5.0, 6.0, 7.0 /)")
write(6,201)b(1),b(2),b(3)
201 format("b(1)=",f6.3," b(2)=",f6.3," b(3)=",f6.3)
write(6,202)b(4),b(5),b(6),b(7)
202 format("b(4)=",f6.3," b(5)=",f6.3," b(6)=",f6.3," b(7)=",f6.3)

write(6,300)
300 format(/,"t300 is (/ 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0 /)")
b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
call daxpy(7,2.0d0,a,1,b,1)
write(6,201)b(1),b(2),b(3)
write(6,202)b(4),b(5),b(6),b(7)

write(6,302)
302 format(/,"t302 is (/ 3.0, 2.0, 3.0, 12.0, 5.0, 6.0, 21.0 /)")
b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
call daxpy(3,2.0d0,a,3,b,3)
write(6,201)b(1),b(2),b(3)
write(6,202)b(4),b(5),b(6),b(7)

write(6,303)
303 format(/,"t303 is (/ 3.0, 2.0, 9.0, 4.0, 15.0, 6.0, 21.0 /)")
b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
call daxpy(4,2.0d0,a,2,b,2)
write(6,201)b(1),b(2),b(3)
write(6,202)b(4),b(5),b(6),b(7)

write(6,305)
305 format(/,"t305 is (/ 3.0, 2.0, 9.0, 4.0, 15.0, 6.0, 7.0 /)")
b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
call daxpy(3,2.0d0,a,2,b,2)
write(6,201)b(1),b(2),b(3)
write(6,202)b(4),b(5),b(6),b(7)

write(6,306)
306 format(/,"t306 is (/ -1.0, 2.0, -1.0, 4.0, -1.0, 6.0, 7.0 /)")
b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
call daxpy(3,-2.0d0,a,1,b,2)
write(6,201)b(1),b(2),b(3)
write(6,202)b(4),b(5),b(6),b(7)

write(6,307)
307 format(/,"t307 is (/ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 /)")

```

```

      b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
      call daxpy(3,0.0d0,a,1,b,2)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)
      stop
      end

```

```
gcc -o daxpyEX daxpyEX.f -lgfortran daxpy.o && ./daxpyEX
```

```

a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000
b(1)= 1.000 b(2)= 2.000 b(3)= 3.000
b(4)= 4.000 b(5)= 5.000 b(6)= 6.000

```

```

t200 is (/ 3.0, 6.0, 9.0, 4.0, 5.0, 6.0, 7.0 /)
b(1)= 3.000 b(2)= 6.000 b(3)= 9.000
b(4)= 4.000 b(5)= 5.000 b(6)= 6.000 b(7)= 7.000

```

```

t300 is (/ 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0 /)
b(1)= 3.000 b(2)= 6.000 b(3)= 9.000
b(4)=12.000 b(5)=15.000 b(6)=18.000 b(7)=21.000

```

```

t302 is (/ 3.0, 2.0, 3.0, 12.0, 5.0, 6.0, 21.0 /)
b(1)= 3.000 b(2)= 2.000 b(3)= 3.000
b(4)=12.000 b(5)= 5.000 b(6)= 6.000 b(7)=21.000

```

```

t303 is (/ 3.0, 2.0, 9.0, 4.0, 15.0, 6.0, 21.0 /)
b(1)= 3.000 b(2)= 2.000 b(3)= 9.000
b(4)= 4.000 b(5)=15.000 b(6)= 6.000 b(7)=21.000

```

```

t305 is (/ 3.0, 2.0, 9.0, 4.0, 15.0, 6.0, 7.0 /)
b(1)= 3.000 b(2)= 2.000 b(3)= 9.000
b(4)= 4.000 b(5)=15.000 b(6)= 6.000 b(7)= 7.000

```

```

t306 is (/ -1.0, 2.0, -1.0, 4.0, -1.0, 6.0, 7.0 /)
b(1)=-1.000 b(2)= 2.000 b(3)=-1.000
b(4)= 4.000 b(5)=-1.000 b(6)= 6.000 b(7)= 7.000

```

```

t307 is (/ 2.0, 2.0, 7.0, 4.0, 11.0, 6.0, 7.0 /)
b(1)= 3.000 b(2)= 2.000 b(3)= 7.000
b(4)= 4.000 b(5)=11.000 b(6)= 6.000 b(7)= 7.000

```

— BLAS 1 daxpy —

```
(declaim (ftype (function (fixnum double-float
```

```

                                (simple-array double-float (*)) fixnum
                                (simple-array double-float (*)) fixnum)
                                (simple-array double-float (*)) daxpy))
(defun daxpy (n da dx incx dy incy)
  ; Tim Daly April 24, 2012
  (declare (type (simple-array double-float) dx dy)
            (type double-float da) (type fixnum incy incx n))
  (let ((maxx (length dx)) (maxy (length dy)))
    (declare (type fixnum maxx maxy))
    (when (and (> n 0) (/= da 0.0)
              (> incx 0) (< (* (1- n) incx) maxx)
              (> incy 0) (< (* (1- n) incy) maxy))
      (if (and (= incx 1) (= incy 1))
          (dotimes (i n)
            (setf (the double-float (svref dy i))
                  (+ (the double-float (svref dy i))
                    (* (the double-float da)
                      (the double-float (svref dx i))))))
          ; non-unit increments
          (let ((ix 0) (iy 0))
            (declare (type fixnum ix iy))
            (when (< incx 0) (setq ix (* (1+ (- n)) incx)))
            (when (< incy 0) (setq iy (* (1+ (- n)) incy)))
            (dotimes (i n)
              (setf (the double-float (svref dy iy))
                    (+ (the double-float (svref dy iy))
                      (* (the double-float da)
                        (the double-float (svref dx ix))))))
              (setq ix (+ ix incx))
              (setq iy (+ iy incy))))))
    dy)

```

— BLAS 1 daxpy lisp test —

```

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(daxpy 3 2.0d0 a 1 b 1)
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(daxpy 7 2.0d0 a 1 b 1)
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(daxpy 8 2.0d0 a 1 b 1)
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(daxpy 3 2.0d0 a 3 b 3)
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(daxpy 4 2.0d0 a 2 b 2)
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))

```



— dcopy.input —

[illegible]

```

--E 3

--S 4 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (4) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 4

--S 5 of 23
dcopy(7,a,1,b,1)
--R
--R
--R (5) [1.,2.,3.,4.,5.,6.,7.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 5

--S 6 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (6) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 6

--S 7 of 23
dcopy(8,a,1,b,1)
--R
--R
--R (7) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 7

--S 8 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (8) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 8

--S 9 of 23
dcopy(3,a,3,b,3)
--R
--R
--R (9) [1.,0.,0.,4.,0.,0.,7.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 9

```

```

--S 10 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (10) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 10

--S 11 of 23
dcopy(4,a,2,b,2)
--R
--R
--R (11) [1.,0.,3.,0.,5.,0.,7.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 11

--S 12 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (12) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 12

--S 13 of 23
dcopy(5,a,2,b,2)
--R
--R
--R (13) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 13

--S 14 of 23
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
--R
--R
--R (14) [0.,0.,0.,0.,0.,0.,0.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 14

--S 15 of 23
dcopy(3,a,2,b,2)
--R
--R
--R (15) [1.,0.,3.,0.,5.,0.,0.]
--R
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 15

--S 16 of 23
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]

```

```

--R
--R
--R   (16)  [1.,2.,3.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 16

--S 17 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (17)  [1.,2.,3.,4.,5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 17

--S 18 of 23
dcopy(3,a,1,b,1)
--R
--R
--R   (18)  [1.,2.,3.,4.,5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 18

--S 19 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (19)  [1.,2.,3.,4.,5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 19

--S 20 of 23
dcopy(3,a,1,b,2)
--R
--R
--R   (20)  [1.,2.,2.,4.,3.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 20

--S 21 of 23
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R
--R   (21)  [1.,2.,3.,4.,5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 21

--S 22 of 23
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
--R
--R

```



```

--R (22) [1.,2.,3.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 22

--S 23 of 23
dcopy(5,a,1,b,1)
--R
--R
--R (23) [1.,2.,3.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 23

)spool
)lisp (bye)

```

— dcopy.help —

```

=====
dcopy examples
=====

```

Assume we have two arrays which we initialize to these values:

```
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] ]
```

```
b:PRIMARR(DFLOAT):=[ [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] ]
```

Note that after each call to dcopy the b array is modified.
The changed value is shown. We reset it after each bcopy but we
do not show that here.

```
dcopy(3,a,1,b,1) ==> [1.,2.,3.,0.,0.,0.,0.]
```

```
dcopy(7,a,1,b,1) ==> [1.,2.,3.,4.,5.,6.,7.]
```

```
dcopy(8,a,1,b,1) ==> [0.,0.,0.,0.,0.,0.,0.]
```

```
dcopy(3,a,3,b,3) ==> [1.,0.,0.,4.,0.,0.,7.]
```

```
dcopy(4,a,2,b,2) ==> [1.,0.,3.,0.,5.,0.,7.]
```

```
dcopy(5,a,2,b,2) ==> [0.,0.,0.,0.,0.,0.,0.]
```

```
dcopy(3,a,2,b,2) ==> [1.,0.,3.,0.,5.,0.,0.]
```

The arrays can be of different lengths:

```
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
```

```
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
```

```
dcopy(3,a,1,b,1) ==> [1.,2.,3.,4.,5.]
```

```
dcopy(3,a,1,b,2) ==> [1.,2.,2.,4.,3.]
```

```
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
```

```
b:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0] ]
```

```
dcopy(5,a,1,b,1) ==> [1.,2.,3.]
```

```
=====
Man Page Details
=====
```

NAME

DCOPY - BLAS level one, copies a double precision vector into another double precision vector

SYNOPSIS

```
SUBROUTINE DCOPY      ( n, x, incx, y, incy )
```

```
      INTEGER          n, incx, incy
```

```
      DOUBLE PRECISION x, y
```

DESCRIPTION

DCOPY copies a double precision vector into another double precision vector. DCOPY copies a vector x, whose length is n to a vector y. incx and incy specify the increment between two consecutive elements of respectively vector x and y.

This routine performs the following vector operation:

$$y \leftarrow x$$

where x and y are double precision vectors.

ARGUMENTS

n INTEGER. (input)
 Number of vector elements to be copied.
 If n <= 0, this routine returns without computation.

x DOUBLE PRECISION, (input)
 Vector from which to copy.

incx INTEGER. (input)

Increment between elements of x.
If incx = 0, the y vector is unchanged

y DOUBLE PRECISION, (output)
 array of dimension (n-1) * |incy| + 1, result vector.

incy INTEGER. (input)
 Increment between elements of y.
 If incy = 0, the y vector is unchanged

— dcopy.f —

```

subroutine dcopy(n,dx,incx,dy,incy)
c
c  copies a vector, x, to a vector, y.
c  uses unrolled loops for increments equal to one.
c  jack dongarra, linpack, 3/11/78.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
  double precision dx(*),dy(*)
  integer i,incx,incy,ix,iy,m,mp1,n
c
  if(n.le.0)return
  if(incx.eq.1.and.incy.eq.1)go to 20
c
c      code for unequal increments or equal increments
c      not equal to 1
c
    ix = 1
    iy = 1
    if(incx.lt.0)ix = (-n+1)*incx + 1
    if(incy.lt.0)iy = (-n+1)*incy + 1
    do 10 i = 1,n
      dy(iy) = dx(ix)
      ix = ix + incx
      iy = iy + incy
10 continue
    return
c
c      code for both increments equal to 1
c
c
c      clean-up loop
c
20 m = mod(n,7)
   if( m .eq. 0 ) go to 40

```

```

do 30 i = 1,m
  dy(i) = dx(i)
30 continue
  if( n .lt. 7 ) return
40 mp1 = m + 1
  do 50 i = mp1,n,7
    dy(i) = dx(i)
    dy(i + 1) = dx(i + 1)
    dy(i + 2) = dx(i + 2)
    dy(i + 3) = dx(i + 3)
    dy(i + 4) = dx(i + 4)
    dy(i + 5) = dx(i + 5)
    dy(i + 6) = dx(i + 6)
50 continue
  return
end

```

Compile with

```

gcc -c dcopy.f
gcc -o dcopyEx dcopyEX.f -lgfortran dcopy.o

```

— dcopy example —

```

program dcopyEX
*   Tim Daly April 27, 2012
*   unit tests for BLAS dcopy
double precision a(7)
double precision b(7)
a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0, 7.0D0 /)
b = (/ 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
write(6,100)a(1),a(2),a(3)
100 format("a(1)=",f6.3," a(2)=",f6.3," a(3)=",f6.3)
write(6,101)a(4),a(5),a(6)
101 format("a(4)=",f6.3," a(5)=",f6.3," a(6)=",f6.3)
write(6,102)b(1),b(2),b(3)
102 format("b(1)=",f6.3," b(2)=",f6.3," b(3)=",f6.3)
write(6,103)b(4),b(5),b(6)
103 format("b(4)=",f6.3," b(5)=",f6.3," b(6)=",f6.3)

call dcopy(3,a,1,b,1)
write(6,200)
200 format(/,"t200 is (/ 1.0 2.0 3.0 0.0 0.0 0.0 0.0 /)")
write(6,201)b(1),b(2),b(3)
201 format("b(1)=",f6.3," b(2)=",f6.3," b(3)=",f6.3)
write(6,202)b(4),b(5),b(6),b(7)

```

```

202  format("b(4)=",f6.3," b(5)=",f6.3," b(6)=",f6.3," b(7)=",f6.3)

      write(6,300)
300  format(/,"t300 is (/ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 /)")
      b = (/ 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
      call dcopy(7,a,1,b,1)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)

      write(6,302)
302  format(/,"t302 is (/ 1.0, 0.0, 0.0, 4.0, 0.0, 0.0, 7.0 /)")
      b = (/ 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
      call dcopy(3,a,3,b,3)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)

      write(6,303)
303  format(/,"t303 is (/ 1.0, 0.0, 3.0, 0.0, 5.0, 0.0, 7.0 /)")
      b = (/ 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
      call dcopy(4,a,2,b,2)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)

      write(6,305)
305  format(/,"t305 is (/ 1.0, 2.0, 3.0, 4.0, 5.0, 0.0, 0.0 /)")
      a = (/ 1.0D0, 2.0D0, 3.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
      b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 0.0D0, 0.0D0 /)
      call dcopy(3,a,1,b,1)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)

      write(6,306)
306  format(/,"t306 is (/ 1.0, 2.0, 2.0, 4.0, 3.0, 0.0, 0.0 /)")
      a = (/ 1.0D0, 2.0D0, 3.0D0, 0.0D0, 0.0D0, 0.0D0, 0.0D0 /)
      b = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 0.0D0, 0.0D0 /)
      call dcopy(3,a,1,b,2)
      write(6,201)b(1),b(2),b(3)
      write(6,202)b(4),b(5),b(6),b(7)

      stop
      end

```

```
gcc -o dcopyEX dcopyEX.f -lgfortran dcopy.o && ./dcopyEX
```

```

a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000
b(1)= 0.000 b(2)= 0.000 b(3)= 0.000
b(4)= 0.000 b(5)= 0.000 b(6)= 0.000

```

```
t200 is (/ 1.0 2.0 3.0 0.0 0.0 0.0 0.0 /)
b(1)= 1.000 b(2)= 2.000 b(3)= 3.000
b(4)= 0.000 b(5)= 0.000 b(6)= 0.000 b(7)= 0.000
```

```
t300 is (/ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0 /)
b(1)= 1.000 b(2)= 2.000 b(3)= 3.000
b(4)= 4.000 b(5)= 5.000 b(6)= 6.000 b(7)= 7.000
```

```
t302 is (/ 1.0, 0.0, 0.0, 4.0, 0.0, 0.0, 7.0 /)
b(1)= 1.000 b(2)= 0.000 b(3)= 0.000
b(4)= 4.000 b(5)= 0.000 b(6)= 0.000 b(7)= 7.000
```

```
t303 is (/ 1.0, 0.0, 3.0, 0.0, 5.0, 0.0, 7.0 /)
b(1)= 1.000 b(2)= 0.000 b(3)= 3.000
b(4)= 0.000 b(5)= 5.000 b(6)= 0.000 b(7)= 7.000
```

```
t305 is (/ 1.0, 2.0, 3.0, 4.0, 5.0, 0.0, 0.0 /)
b(1)= 1.000 b(2)= 2.000 b(3)= 3.000
b(4)= 4.000 b(5)= 5.000 b(6)= 0.000 b(7)= 0.000
```

```
t306 is (/ 1.0, 2.0, 2.0, 4.0, 3.0, 0.0, 0.0 /)
b(1)= 1.000 b(2)= 2.000 b(3)= 2.000
b(4)= 4.000 b(5)= 3.000 b(6)= 0.000 b(7)= 0.000
```

— BLAS 1 dcopy —

```
(declaim (ftype (function (fixnum (simple-array double-float (*)) fixnum
                             (simple-array double-float (*)) fixnum)
                    (simple-array double-float (*))) dcopy))
(defun dcopy (n dx incx dy incy)
  ; Tim Daly April 27, 2012
  (declare (type (simple-array double-float) dy dx)
            (type fixnum incy incx n))
  (let ((maxx (length dx)) (maxy (length dy)))
    (declare (type fixnum maxx maxy))
    (when (and (> n 0)
                (> incx 0) (< (* (1- n) incx) maxx)
                (> incy 0) (< (* (1- n) incy) maxy))
      (if (and (= incx 1) (= incy 1))
          ; unit increments
          (dotimes (i n)
            (setf (the double-float (svref dy i)) (the double-float (svref dx i))))
          ; non-unit increments
          (let ((ix 0) (iy 0))
            (declare (type fixnum ix iy))
            (when (< incx 0) (setq ix (* (1+ (- n)) incx)))
            (when (< incy 0) (setq ix (* (1+ (- n)) incy))))
```

```

(dotimes (i n)
  (setf (the double-float (svref dy iy)) (the double-float (svref dx ix)))
  (setq ix (+ ix incx))
  (setq iy (+ iy incy))))))
dy)

```

— BLAS 1 dcopy lisp test —

```

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0 7.0d0))
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 3 a 1 b 1)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 7 a 1 b 1)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 8 a 1 b 1)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 3 a 3 b 3)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 4 a 2 b 2)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 5 a 2 b 2)
(setq b (vector 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0))
(dcopy 3 a 2 b 2)
(setq a (vector 1.0d0 2.0d0 3.0d0))
(setq b (vector 0.0d0 0.0d0 3.0d0 4.0d0 5.0d0))
(dcopy 3 a 1 b 1)
b
(setq b (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
(dcopy 3 a 1 b 2)
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
(setq b (vector 1.0d0 2.0d0 3.0d0))
(dcopy 5 a 1 b 1)

```

ddot BLAS

— ddot.input —

```

)set break resume
)sys rm -f ddot.output
)spool ddot.output
)set message test on

```

```

)set message auto off
)clear all

--S 1 of 9
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
--R
--R (1) [1.,2.,3.,4.,5.]
--R
--R Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 9
b:PRIMARR(DFLOAT):=[ [ 5.0, 6.0, 7.0, 8.0, 9.0] ]
--R
--R (2) [5.,6.,7.,8.,9.]
--R
--R Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 9
ddot(0,a,1,b,1)
--R
--R (3) 0.
--R
--R Type: DoubleFloat
--E 3

--S 4 of 9
ddot(3,a,1,b,1)
--R
--R (4) 38.
--R
--R Type: DoubleFloat
--E 4

--S 5 of 9
ddot(3,a,1,b,2)
--R
--R (5) 46.
--R
--R Type: DoubleFloat
--E 5

--S 6 of 9
ddot(3,a,2,b,1)
--R
--R (6) 58.
--R
--R Type: DoubleFloat
--E 6

--S 7 of 9
ddot(3,a,1,b,-2)
--R
--R (7) 38.
--R
--R Type: DoubleFloat

```



```
--E 7
```

```
--S 8 of 9
```

```
ddot(3,a,-2,b,1)
```

```
--R
```

```
--R (8) 50.
```

```
--R
```

Type: DoubleFloat

```
--E 8
```

```
--S 9 of 9
```

```
ddot(3,a,-2,b,-2)
```

```
--R
```

```
--R (9) 71.
```

```
--R
```

Type: DoubleFloat

```
--E 9
```

```
)spool
```

```
)lisp (bye)
```

— ddot.help —

```
=====
ddot examples
=====
```

```
a:PRIMARR(DFLOAT):=[ [ 1.0, 2.0, 3.0, 4.0, 5.0] ]
```

```
[1.,2.,3.,4.,5.]
```

```
b:PRIMARR(DFLOAT):=[ [ 5.0, 6.0, 7.0, 8.0, 9.0] ]
```

```
[5.,6.,7.,8.,9.]
```

```
ddot(0,a,1,b,1)
```

```
0.
```

```
ddot(3,a,1,b,1)
```

```
38. ( = 1.0*5.0 + 2.0*6.0 + 3.0*7.0 )
```

```
ddot(3,a,1,b,2)
```

```
46. ( = 1.0*5.0 + 2.0*7.0 + 3.0*9.0 )
```

```
ddot(3,a,2,b,1)
```

```

58. ( = 1.0*5.0 + 3.0*6.0 + 5.0*7.0 )

ddot(3,a,1,b,-2)

38. ( = 1.0*9.0 + 2.0*7.0 + 3.0*5.0 )

ddot(3,a,-2,b,1)

50. ( = 5.0*5.0 + 3.0*6.0 + 1.0*7.0 )

ddot(3,a,-2,b,-2)

71. ( = 5.0*9.0 + 3.0*7.0 + 1.0*5.0 )

```

```

=====
Man Page Details
=====

```

NAME

DDOT - BLAS level one, computes a dot product (inner product) of two double precision vectors

SYNOPSIS

DOUBLE PRECISION FUNCTION DDOT (n, x, incx, y, incy)

INTEGER n, incx, incy

DOUBLE PRECISION x, y

DESCRIPTION

DDOT computes a dot product of two double precision vectors (1 double precision inner product).

2

This routine performs the following vector operation:

$$\text{DDOT} \leftarrow (\text{transpose of } x) * y = \sum_{i=1}^n x(i)*y(i)$$

where x and y are double precision vectors.

If n <= 0, DDOT is set to 0.

ARGUMENTS

n INTEGER. (input)
Number of elements in each vector.

x DOUBLE PRECISION. (input)

Array of dimension $(n-1) * |incx| + 1$.
 Array x contains the first vector operand.

incx INTEGER. (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

y DOUBLE PRECISION, (input)
 Array of dimension $(n-1) * |incy| + 1$.
 Array y contains the second vector operand.

incy INTEGER. (input)
 Increment between elements of y. If incy = 0, the results will
 be unpredictable.

RETURN VALUES

DDOT DOUBLE PRECISION. Result (dot product). (output)

NOTES

When working backward ($incx < 0$ or $incy < 0$), each routine starts at
 the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

$y(1-incy * (n-1)), y(1-incy * (n-2)), \dots, y(1)$

— ddot.f —

```
double precision function ddot(n,dx,incx,dy,incy)
c
c  forms the dot product of two vectors.
c  uses unrolled loops for increments equal to one.
c  jack dongarra, linpack, 3/11/78.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
double precision dx(*),dy(*),dtemp
integer i,incx,incy,ix,iy,m,mp1,n
c
ddot = 0.0d0
dtemp = 0.0d0
if(n.le.0)return
if(incx.eq.1.and.incy.eq.1)go to 20
c
c      code for unequal increments or equal increments
c      not equal to 1
c
```

```

ix = 1
iy = 1
if(incx.lt.0)ix = (-n+1)*incx + 1
if(incy.lt.0)iy = (-n+1)*incy + 1
do 10 i = 1,n
    dtemp = dtemp + dx(ix)*dy(iy)
    ix = ix + incx
    iy = iy + incy
10 continue
ddot = dtemp
return
c
c      code for both increments equal to 1
c
c
c      clean-up loop
c
20 m = mod(n,5)
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
       dtemp = dtemp + dx(i)*dy(i)
30 continue
   if( n .lt. 5 ) go to 60
40 mp1 = m + 1
   do 50 i = mp1,n,5
       dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
*      dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
50 continue
60 ddot = dtemp
   return
end

```

Compile with

```

gcc -c ddot.f
gcc -o ddotEx ddotEX.f -lgfortran ddot.o

```

— ddot example —

```

program ddotEX
*      Tim Daly April 27, 2012
*      unit tests for BLAS ddot
double precision a(5)
double precision b(5)
double precision c
a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0 /)

```

```

        b = (/ 5.0D0, 6.0D0, 7.0D0, 8.0D0, 9.0D0 /)
        write(6,100)a(1),a(2),a(3),a(4),a(5)
100    format("a=(",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3,")")
        write(6,101)b(1),b(2),b(3),b(4),b(5)
101    format("b=(",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3,")")

* handle 0 elements
    c=ddot(0,a,1,b,1)
    write(6,200)c
200    format(/,"t200 is 0.0",/,"c=",f6.3)

* handle (1,2,3) * (5,6,7)
    c=ddot(3,a,1,b,1)
    write(6,201)c
201    format(/,"t201 is 38.0",/,"c=",f6.3)

* handle increment = 2 in b (1,2,3) * (5,7,9)
    c=ddot(3,a,1,b,2)
    write(6,202)c
202    format(/,"t202 is 46.0",/,"c=",f6.3)

* handle increment = 2 in a (1,3,5) * (5,6,7)
    c=ddot(3,a,2,b,1)
    write(6,203)c
203    format(/,"t203 is 58.0",/,"c=",f6.3)

* handle increment = -2 in b (1,2,3) * (9,7,5)
    c=ddot(3,a,1,b,-2)
    write(6,204)c
204    format(/,"t204 is 38.0",/,"c=",f6.3)

* handle increment = -2 in a (5,3,1) * (5,6,7)
    c=ddot(3,a,-2,b,1)
    write(6,205)c
205    format(/,"t205 is 50.0",/,"c=",f6.3)

* handle increment = -2 in a and b(5,3,1) * (9,7,5)
    c=ddot(3,a,-2,b,-2)
    write(6,206)c
206    format(/,"t206 is 71.0",/,"c=",f6.3)

    stop
    end

_____

a=(/ 1.000 2.000 3.000 4.000 5.000/)
a=(/ 5.000 6.000 7.000 8.000 9.000/)

t200 is 0.0

```

```

c= 0.000

t201 is 28.0
c=38.000

t202 is 46.0
c=46.000

t203 is 58.0
c=58.000

t204 is 38.0
c=38.000

t205 is 50.0
c=50.000

t206 is 71.0
c=71.000

```

— BLAS 1 ddot —

```

(declaim (ftype (function (fixnum (simple-array double-float (*)) fixnum
                                (simple-array double-float (*)) fixnum)
                        double-float) ddot))
(defun ddot (n dx incx dy incy)
  ; Tim Daly April 27, 2012
  (declare (optimize (speed 3) (safety 0))
    (type (simple-array double-float (*)) dx dy)
    (type fixnum incy incx n))
  (let ((ix 0) (iy 0) (ddot 0.0d0))
    (declare (type (double-float) ddot) (type fixnum iy ix))
    (when (> n 0)
      (when (< incx 0)
        (setf ix (the fixnum (*
          (the fixnum (- 1 (the fixnum n)))
          (the fixnum incx)))))
      (when (< incy 0)
        (setf iy (the fixnum (*
          (the fixnum (- 1 (the fixnum n)))
          (the fixnum incy)))))
      (do ((i 0 (1+ i)) (x ix (+ x incx)) (y iy (+ y incy)))
          ((= i n))
        (setf ddot
          (the double-float (+ ddot
            (the double-float (* (aref dx x) (aref dy y)))))))
      ddot))

```

```
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
(setq b (vector 5.0d0 6.0d0 7.0d0 8.0d0 9.0d0))
(= 0.0d0 (ddot 0 a 1 b 1))
(= 38.0d0 (ddot 3 a 1 b 1))
(= 46.0d0 (ddot 3 a 1 b 2))
(= 58.0d0 (ddot 3 a 2 b 1))
(= 38.0d0 (ddot 3 a 1 b -2))
(= 50.0d0 (ddot 3 a -2 b 1))
(= 71.0d0 (ddot 3 a -2 b -2))
```

— dnrm2.input —

[illegible]

```
=====
dnrm2 examples
=====
```

Given the vector [3.0, -4.0, 5.0, -7.0, 9.0]

```
=====
Man Page Details
=====
```

```
SYNOPSIS
      DOUBLE PRECISION FUNCTION DNRM2 ( n, x, incx )

      INTEGER                                n, incx
```


DOUBLE PRECISION x

DESCRIPTION

DNRM2 computes the Euclidean (L2) norm of a double precision real vector, as follows:

$$\text{DNRM2} \leftarrow \frac{\|x\|}{2}$$

where x is a double precision real vector.

ARGUMENTS

n INTEGER. (input)
 Number of elements in the operand vector.

x DOUBLE PRECISION. (input)
 Array of dimension (n-1) * |incx| + 1.
 Array x contains the operand vector.

incx INTEGER. (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

RETURN VALUES

DNRM2 DOUBLE PRECISION. Result (Euclidean norm). (output)
 If n <= 0, DNRM2 is set to 0d0.

NOTES

When working backward (incx < 0), each routine starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

— dnrnm2.f —

```
DOUBLE PRECISION FUNCTION DNRM2 ( N, X, INCX )
*   .. Scalar Arguments ..
INTEGER                                INCX, N
*   .. Array Arguments ..
DOUBLE PRECISION                      X( * )
*   ..
*
* DNRM2 returns the euclidean norm of a vector via the function
* name, so that
*
```

```

*      DNRM2 := sqrt( x'*x )
*
*
*
*
*      -- This version written on 25-October-1982.
*      Modified on 14-October-1993 to inline the call to DCLASSQ.
*      Sven Hammarling, Nag Ltd.
*
*
*      .. Parameters ..
      DOUBLE PRECISION      ONE          , ZERO
      PARAMETER              ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*      .. Local Scalars ..
      INTEGER                IX
      DOUBLE PRECISION      ABSXI, NORM, SCALE, SSQ
*      .. Intrinsic Functions ..
      INTRINSIC              ABS, SQRT
*
*      .. Executable Statements ..
      IF( N.LT.1 .OR. INCX.LT.1 )THEN
          NORM = ZERO
      ELSE IF( N.EQ.1 )THEN
          NORM = ABS( X( 1 ) )
      ELSE
          SCALE = ZERO
          SSQ = ONE
*          The following loop is equivalent to this call to the LAPACK
*          auxiliary routine:
*          CALL DCLASSQ( N, X, INCX, SCALE, SSQ )
*
          DO 10, IX = 1, 1 + ( N - 1 )*INCX, INCX
              IF( X( IX ).NE.ZERO )THEN
                  ABSXI = ABS( X( IX ) )
                  IF( SCALE.LT.ABSXI )THEN
                      SSQ = ONE + SSQ*( SCALE/ABSXI )**2
                      SCALE = ABSXI
                  ELSE
                      SSQ = SSQ + ( ABSXI/SCALE )**2
                  END IF
              END IF
          10 CONTINUE
          NORM = SCALE * SQRT( SSQ )
      END IF
*
*      DNRM2 = NORM
      RETURN
*
*      End of DNRM2.
*
      END

```

Compile with

```
gcc -c dnrnm2.f
gcc -o dnrnm2EX dnrnm2EX.f -lgfortran dnrnm2.o
```

— dnrnm2 example —

```

      program dnrnm2EX
*      Tim Daly April 28, 2012
*      unit tests for BLAS dnrnm2
      double precision a(5)
      double precision c
      a = (/ 3.0D0, -4.0D0, 5.0D0, -7.0D0, 9.0D0/)
      write(6,100)a(1),a(2),a(3)
100   format("a=(",f6.3," ",f6.3," ",f6.3"/)")

      c=dnrm2(3,a,1)
      write(6,200)c
200   format(/,"t200 is sqrt(50.0)=7.071",/,"c=",f6.3)

      c=dnrm2(5,a,1)
      write(6,201)c
201   format(/,"t201 is sqrt(180.0)=13.416",/,"c=",f6.3)

      c=dnrm2(3,a,2)
      write(6,202)c
202   format(/,"t202 is sqrt(115.0)=10.724",/,"c=",f9.3)

      stop
      end

```

```
gcc -o dnrnm2EX dnrnm2EX.f -lgfortran dnrnm2.o && ./dnrm2EX
a=(/ 3.000 -4.000  5.000/)
```

```
t200 is sqrt(50.0)=7.071
c= 7.071
```

```
t201 is sqrt(180.0)=13.416
c=13.416
```

```
t202 is sqrt(115.0)=10.724
c= 10.724
```

— BLAS 1 dnorm2 —

```

(declaim (ftype (function (fixnum (simple-array double-float (*)) fixnum)
                        double-float) dnorm2))
(defun dnorm2 (n x incx)
  ; Tim Daly April 28, 2012
  (declare (type (simple-array double-float (*)) x) (type fixnum incx n))
  (let ((absxi 0.0d0) (norm 0.0d0) (scale 0.0d0) (ssq 0.0d0)
        (limit 0) (t1 0.0d0))
    (declare (type fixnum limit) (type double-float absxi norm scale ssq t1))
    (cond
      ((or (< n 1) (< incx 1)) (setf norm 0.0d0))
      ((= n 1) (setf norm (abs (aref x 0)))))
    (t
     (setf limit (min (* (- n 1) incx) (1- (length x))))
     (do ((ix 0 (+ ix incx)))
       ((> ix limit) nil)
       (unless (= 0.0d0 (the double-float (aref x ix)))
         (setf absxi (the double-float (abs (the double-float (aref x ix)))))
         (cond
          ((< scale absxi)
           (setf t1 (the double-float
                     (/ (the double-float scale) (the double-float absxi)))))
          (t
           (setf ssq
                 (the double-float
                  (+ 1.0d0
                    (the double-float
                     (* (the double-float ssq)
                       (the double-float
                        (* (the double-float t1) (the double-float t1))))))))
           (setf scale absxi)))
       (t
        (setf t1 (the double-float
                  (/ (the double-float absxi) (the double-float scale)))))
        (setf ssq
              (the double-float
               (+ (the double-float ssq)
                 (the double-float
                  (* (the double-float t1) (the double-float t1))))))))
     (setf norm (the double-float (* scale (the double-float (sqrt ssq)))))
     norm))

```

— BLAS 1 dnorm2 lisp test —

```

(setq a (vector 3.0d0 -4.0d0 5.0d0 -7.0d0 9.0d0))

```

```
; sqrt(50.0) = 7.0710678118654755
(dnrm2 3 a 1)
; sqrt(180.0) = 13.416407864998739
(dnrm2 5 a 1)
; sqrt(115.0) = 10.72380529476361
(dnrm2 3 a 2)
```

drotg BLAS

— drotg.input —

```
)set break resume
)sys rm -f drotg.output
)spool drotg.output
)set message test on
)set message auto off
)clear all

--S 1 of 8
a:MATRIX(DFLOAT):=[[6,5,0],[5,1,4],[0,4,3]]
--R
--R      +6.  5.  0.+
--R      |      |
--R  (1) |5.  1.  4.|
--R      |      |
--R      +0.  4.  3.+
--R
--R                                          Type: Matrix(DoubleFloat)
--E 1

--S 2 of 8
t1:=drotg(elt(a,1,1),elt(a,1,2),0.0,0.0)
--R
--R  (2)
--R  [7.810249675906654, 0.64018439966447993, 0.76822127959737585,
--R   0.64018439966447993]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 8
g1:MATRIX(DFLOAT):=[[elt(t1,2), elt(t1,3),0.0],_
                    [-elt(t1,3),elt(t1,2),0.0],_
                    [0.0,      0.0,      1.0]]
--R
--R
--R      + 0.76822127959737585  0.64018439966447993  0.+
```

[illegible]

```

--E 7

--S 8 of 8
q:=transpose(g1)*transpose(g2)
--R
--R      +0.76822127959737585   0.33265417936007158   0.54697098874441952 +
--R      |
--R      (8) |0.64018439966447993 - 0.39918501523208583 - 0.65636518649330344|
--R      |
--R      +      0.      0.85439599751428896 - 0.51962243930719854+
--R                                          Type: Matrix(DoubleFloat)
--E 8

)spool
)lisp (bye)

```

— drotg.help —

=====

drotg examples

=====

A Givens rotation is a rotation in the plane spanned by two coordinate axes, named after Wallace Givens. [REF-Wiki3]

A Givens rotation is represented by a matrix of the form

$$G(i,j,\theta) = \begin{array}{c} \begin{array}{cc} + - & - + \\ | 1 & \dots & 0 & \dots & 0 & \dots & 0 & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | 0 & \dots & c & \dots & -s & \dots & 0 & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | 0 & \dots & s & \dots & c & \dots & 0 & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | . & . & . & . & . & . & . & | \\ | 0 & \dots & 0 & \dots & 0 & \dots & 1 & | \\ + - & - + \end{array} \end{array}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ appear at the intersections of the i th, j th rows and columns. The non-zero elements of a Givens matrix are given by:

```

g    = 1 for k != i,j
  kk
g    = c
  ii
g    = c          (sign of sine switches for j>i)
  jj
g    = -s
  ji
g    = s  for i > j
  ij

```

The product $G(i,j,\theta)*x$ represents the counterclockwise rotation of the vector in the (i,j) plane of θ radians.

The main use of Givens rotations in numerical linear algebra is to introduce zeros in vectors or matrices. This effect can be employed for computing the QR decomposition of a matrix. One advantage over the Householder transformations is that they can easily be parallelized, and another is that often for very sparse matrices they have a lower operation count.

When a Givens rotation matrix, $G(i,k,\theta)$ multiplies another matrix A from the left, GA , only rows i and j of A are affected. Thus we restrict attention to the following problem. Given a and b , find $c=\cos(\theta)$ and $s=\sin(\theta)$ such that

$$\begin{array}{rcccl}
 + & - & + & - & + & - \\
 | & c & -s & | & | & a & | & | & r & | \\
 | & & & | & | & & | & = & | & | \\
 | & s & c & | & | & b & | & | & 0 & | \\
 + & - & + & - & + & -
 \end{array}$$

The solution is

```

r = sqrt(a^2+b^2)
c = a/r
s = -b/r

```

However, the computation for r may overflow or underflow. An alternative formulation avoiding this problem [REF-GC96, p5.1.8] is implemented as the `hypot` function in many programming languages.

As Anderson [REF-And00] discovered in improving LAPACK, a previously overlooked numerical consideration is continuity. To achieve this we require r to be positive.

```

if (b = 0) then { c = copysign(1,a); s=0; r=abs(a) }
else if (a = 0) then { c = 0; s = copysign(1,b); r = abs(b) }
else if (abs(b) > abs(a)) then
  t = a/b

```



```

    u = copysign(sqrt(1+t*t),b)
    s = 1/u
    c = s*t
    r = b*u
else
    t = b/a
    u = copysign(sqrt(1+t*t),a)
    c = 1/u
    s = c*t
    r = a*u

```

copysign can be implemented as $x*\text{sgn}(y)$ using the sign function.

=====

Man Page Details

=====

NAME

DGROTG - BLAS level one rotation subroutines

SYNOPSIS

SUBROUTINE DROTG (a, b, c, s)

DOUBLE PRECISION a, b, c, s

DESCRIPTION

DROTG computes the elements of a Givens plane rotation matrix such that:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

where $r = \pm \sqrt{a^2 + b^2}$ and $c^2 + s^2 = 1$.

The Givens plane rotation can be used to introduce zero elements into a matrix selectively.

ARGUMENTS

a (input and output) DOUBLE PRECISION

First vector component. On input, the first component of the vector to be rotated. On output, a is overwritten by r, the first component of the vector in the rotated coordinate system where:

```

r = sign(sqrt(a**2 + b**2),a), if |a| > |b|

r = sign(sqrt(a**2 + b**2),b), if |a| <= |b|

b      (input and output) DOUBLE PRECISION
      Second vector component.

      On input, the second component of the vector to be rotated.  On
      output, b contains z, where:

      z=s      if |a| > |b|
      z=1/c    if |a| <= |b| and c != 0 and r != 0
      z=1      if |a| <= |b| and c = 0 and r != 0
      z=0      if r = 0

c      (output) DOUBLE PRECISION
      Cosine of the angle of rotation:

      c = a/r  if r != 0
      c = 1    if r = 0

s      (output) DOUBLE PRECISION
      Sine of the angle of rotation:

      s = b/r  if r != 0
      s = 0    if r = 0

```

NOTE

The value of z , returned in b by DROTG, gives a compact representation of the rotation matrix, which can be used later to reconstruct c and s as in the following example:

```

IF (B .EQ. 1. ) THEN
  C = 0.
  S = 1.
ELSEIF( ABS( B) .LT. 1) THEN
  C = SQRT( 1. - B * B)
  S = B
ELSE
  C = 1. / B
  S = SQRT( 1 - C * C)
ENDIF

```

Double precision. Computes plane rotation. Arguments are:

- da - double-float

- db - double-float
- c - double-float
- s - double-float

Returns multiple values where:

- 1 da - double-float
- 2 db - double-float
- 3 c - double-float
- 4 s - double-float

— drotg.f —

```

subroutine drotg(da,db,c,s)
c
c   construct givens plane rotation.
c   jack dongarra, linpack, 3/11/78.
c
c   double precision da,db,c,s,roe,scale,r,z
c
    roe = db
    if( dabs(da) .gt. dabs(db) ) roe = da
    scale = dabs(da) + dabs(db)
    if( scale .ne. 0.0d0 ) go to 10
        c = 1.0d0
        s = 0.0d0
        r = 0.0d0
        z = 0.0d0
        go to 20
10 r = scale*dsqrt((da/scale)**2 + (db/scale)**2)
    r = dsign(1.0d0,roe)*r
    c = da/r
    s = db/r
    z = 1.0d0
    if( dabs(da) .gt. dabs(db) ) z = s
    if( dabs(db) .ge. dabs(da) .and. c .ne. 0.0d0 ) z = 1.0d0/c
20 da = r
    db = z
    return
end

```

— drotg example —

```

program drotgEX
*   Tim Daly May 2, 2012
*   unit tests for BLAS drotg
double precision a11,a12,a13,a21,a22,a23,a31,a32,a33
double precision b11,b12,b13,b21,b22,b23,b31,b32,b33
double precision c11,c12,c13,c21,c22,c23,c31,c32,c33
double precision d11,d12,d13,d21,d22,d23,d31,d32,d33
double precision e11,e12,e13,e21,e22,e23,e31,e32,e33
double precision f11,f12,f13,f21,f22,f23,f31,f32,f33
double precision a,b,c,s
a11=6.0d0
a12=5.0d0
a13=0.0d0
a21=5.0d0
a22=1.0d0
a23=4.0d0
a31=0.0d0
a32=4.0d0
a33=3.0d0
write(6,10)
write(6,20)
write(6,30)a11,a12,a13
write(6,30)a21,a22,a23
write(6,30)a31,a32,a33
write(6,20)
10  format("A=");
20  format("  +-                -+")
30  format(" | ",f6.3," ",f6.3," ",f6.3," |")

a=a11
b=a21
c=0.0d0
s=0.0d0
write(6,100)a,b,c,s
100 format(/,"a=",f6.3," b=",f6.3," c=",f6.3," s=",f6.3);

call drotg(a,b,c,s)
write(6,100)a,b,c,s

b11=c
b12=s
b13=0.0d0
b21=-s
b22=c
b23=0.0d0
b31=0.0d0
b32=0.0d0
b33=1.0d0
write(6,11)
11  format(/,"G1=")

```

```

write(6,20)
write(6,30)b11,b12,b13
write(6,30)b21,b22,b23
write(6,30)b31,b32,b33
write(6,20)

c11=b11*a11+b12*a21+b13*a31
c21=b21*a11+b22*a21+b23*a31
c31=b31*a11+b32*a21+b33*a31
c12=b11*a12+b12*a22+b13*a32
c22=b21*a12+b22*a22+b23*a32
c32=b31*a12+b32*a22+b33*a32
c13=b11*a13+b12*a23+b13*a33
c23=b21*a13+b22*a23+b23*a33
c33=b31*a13+b32*a23+b33*a33
write(6,12)
12  format(/,"G1*A=")
write(6,20)
write(6,30)c11,c12,c13
write(6,30)c21,c22,c23
write(6,30)c31,c32,c33
write(6,20)

a=c22
b=c32
c=0.0d0
s=0.0d0
write(6,100)a,b,c,s

call drotg(a,b,c,s)
write(6,100)a,b,c,s

d11=1.0d0
d12=0.0d0
d13=0.0d0
d21=0.0d0
d22=c
d23=s
d31=0.0d0
d32=-s
d33=c
write(6,13)
13  format(/,"G2=")
write(6,20)
write(6,30)d11,d12,d13
write(6,30)d21,d22,d23
write(6,30)d31,d32,d33
write(6,20)

e11=d11*c11+d12*c21+d13*c31

```

```

e21=d21*c11+d22*c21+d23*c31
e31=d31*c11+d32*c21+d33*c31
e12=d11*c12+d12*c22+d13*c32
e22=d21*c12+d22*c22+d23*c32
e32=d31*c12+d32*c22+d33*c32
e13=d11*c13+d12*c23+d13*c33
e23=d21*c13+d22*c23+d23*c33
e33=d31*c13+d32*c23+d33*c33
write(6,14)
14  format(/,"G2*G1*A=")
write(6,20)
write(6,30)e11,e12,e13
write(6,30)e21,e22,e23
write(6,30)e31,e32,e33
write(6,20)

f11=b31*d13 + b21*d12 + b11*d11
f12=b31*d23 + b21*d22 + b11*d21
f13=b31*d33 + b21*d32 + b11*d31
f21=b32*d13 + b22*d12 + b12*d11
f22=b32*d23 + b22*d22 + b12*d21
f23=b32*d33 + b22*d32 + b12*d31
f31=b33*d13 + b23*d12 + b13*d11
f32=b33*d23 + b23*d22 + b13*d21
f33=b33*d33 + b23*d32 + b13*d31
write(6,15)
15  format(/,"Q=transpose(G1)*transpose(G2)")
write(6,20)
write(6,30)f11,f12,f13
write(6,30)f21,f22,f23
write(6,30)f31,f32,f33
write(6,20)

stop
end

```

```
gcc -o drotgEX drotgEX.f -lgfortran drotg.o && ./drotgEX
```

```

A=
+-              +-
| 6.000  5.000  0.000 |
| 5.000  1.000  4.000 |
| 0.000  4.000  3.000 |
+-              +-

a= 6.000 b= 5.000 c= 0.000 s= 0.000

a= 7.810 b= 0.640 c= 0.768 s= 0.640

```

```

G1=
+-              +-
|  0.768  0.640  0.000  |
| -0.640  0.768  0.000  |
|  0.000  0.000  1.000  |
+-              +-

G1*A=
+-              +-
|  7.810  4.481  2.561  |
| -0.000 -2.433  3.073  |
|  0.000  4.000  3.000  |
+-              +-

a=-2.433 b= 4.000 c= 0.000 s= 0.000

a= 4.682 b=-1.924 c=-0.520 s= 0.854

G2=
+-              +-
|  1.000  0.000  0.000  |
|  0.000 -0.520  0.854  |
|  0.000 -0.854 -0.520  |
+-              +-

G2*G1*A=
+-              +-
|  7.810  4.481  2.561  |
|  0.000  4.682  0.966  |
|  0.000  0.000 -4.184  |
+-              +-

Q=transpose(G1)*transpose(G2)
+-              +-
|  0.768  0.333  0.547  |
|  0.640 -0.399 -0.656  |
|  0.000  0.854 -0.520  |
+-              +-

```

— BLAS 1 drotg —

```

(defun drotg (da db c s)
  ; Tim Daly May 2, 2012
  (declare (type (double-float) s c db da))
  (let ((roe 0.0d0) (scale 0.0d0) (r 0.0d0) (z 0.0d0))
    (declare (type (double-float) z r scale roe))
    (setf roe db)

```

```
(load "drotg.lisp")
(defun m (m i j) (svref (svref m (1- i)) (1- j)))
(defun matprint (mat)
  (format t "+-~%" )
  (format t "| ~6,3f ~6,3f ~6,3f |~%" (m mat 1 1) (m mat 1 2) (m mat 1 3))
  (format t "| ~6,3f ~6,3f ~6,3f |~%" (m mat 2 1) (m mat 2 2) (m mat 2 3))
  (format t "| ~6,3f ~6,3f ~6,3f |~%" (m mat 3 1) (m mat 3 2) (m mat 3 3))
  (format t "+-~%" ))
(defun matmult (a b)
  (vector
    (vector (+ (* (m a 1 1) (m b 1 1))
               (* (m a 1 2) (m b 2 1))
               (* (m a 1 3) (m b 3 1))))
    (+ (* (m a 1 1) (m b 1 2))
       (* (m a 1 2) (m b 2 2))
       (* (m a 1 3) (m b 3 2))))
```



```

      (+ (* (m a 1 1) (m b 1 3))
         (* (m a 1 2) (m b 2 3))
         (* (m a 1 3) (m b 3 3))))
(vector (+ (* (m a 2 1) (m b 1 1))
           (* (m a 2 2) (m b 2 1))
           (* (m a 2 3) (m b 3 1))))
      (+ (* (m a 2 1) (m b 1 2))
         (* (m a 2 2) (m b 2 2))
         (* (m a 2 3) (m b 3 2)))
      (+ (* (m a 2 1) (m b 1 3))
         (* (m a 2 2) (m b 2 3))
         (* (m a 2 3) (m b 3 3))))
(vector (+ (* (m a 3 1) (m b 1 1))
           (* (m a 3 2) (m b 2 1))
           (* (m a 3 3) (m b 3 1))))
      (+ (* (m a 3 1) (m b 1 2))
         (* (m a 3 2) (m b 2 2))
         (* (m a 3 3) (m b 3 2)))
      (+ (* (m a 3 1) (m b 1 3))
         (* (m a 3 2) (m b 2 3))
         (* (m a 3 3) (m b 3 3))))))
(defun transpose (mat)
  (vector (vector (m mat 1 1) (m mat 1 2) (m mat 1 3))
          (vector (m mat 2 1) (m mat 2 2) (m mat 2 3))
          (vector (m mat 3 1) (m mat 3 2) (m mat 3 3))))
(setq a #(#(6.0d0 5.0d0 0.0d0)
           #(5.0d0 1.0d0 4.0d0)
           #(0.0d0 4.0d0 3.0d0)))
(multiple-value-setq (x y c s) (drotg (m a 1 1) (m a 2 1) 0.0d0 0.0d0))
(list x y c s)
(setq g1 (vector (vector c      s 0.0d0)
                 (vector (- s)  c 0.0d0)
                 (vector 0.0d0 0.0d0 1.0d0)))
(matprint (setq g1a (matmult g1 a)))
(multiple-value-setq (xx yy cc ss) (drotg (m g1a 2 2) (m g1a 3 2) 0.0d0 0.0d0))
(list xx yy cc ss)
(matprint (setq g2 (vector (vector 1.0d0 0.0d0 0.0d0) (vector 0.0d0 cc ss) (vector 0.0d0 (- ss) cc))))
(matprint (setq g2 (matmult g1 a)))

```

drot BLAS

— drot.input —

```
)set break resume
```

```

)sys rm -f drot.output
)spool drot.output
)set message test on
)set message auto off
)clear all

--S 1 of 17
dx:PRIMARR(DFLOAT):=[[6,0, 1.0, 4.0, -1.0, -1.0]]
--R
--R (1) [6.,0.,1.,4.,- 1.,- 1.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 17
dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]
--R
--R (2) [5.,1.,- 4.,4.,- 4.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
--R
--R (3)
--R [
--R [7.778174591, 0.70710678100000002, - 2.1213203429999998,
--R 5.6568542480000001, - 3.5355339050000003, - 1.]
--R ,
--R [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R - 2.1213203429999998]
--R ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 3

--S 4 of 17
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R (4)
--R [
--R [7.778174591, 0.70710678100000002, - 2.1213203429999998,
--R 5.6568542480000001, - 3.5355339050000003, - 1.]
--R ,
--R [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R - 2.1213203429999998]
--R ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 4

```

```

--S 5 of 17
drot(5,dx,1,dy,1,0.707106781,-0.707106781) -- rotate by -45 degrees
--R
--R (5)
--R [
--R [5.9999999968341839, 7.8496237287950521E-17, 0.99999999947236384,
--R 3.9999999978894558, - 0.99999999947236451, - 1.]
--R ,
--R [4.9999999973618188, 0.99999999947236384, - 3.9999999978894558,
--R 3.9999999978894554, - 3.9999999978894554]
--R ]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 5

--S 6 of 17
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R (6)
--R [
--R [5.9999999968341839, 7.8496237287950521E-17, 0.99999999947236384,
--R 3.9999999978894558, - 0.99999999947236451, - 1.]
--R ,
--R [4.9999999973618188, 0.99999999947236384, - 3.9999999978894558,
--R 3.9999999978894554, - 3.9999999978894554]
--R ]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 6

--S 7 of 17
dx:PRIMARR(DFLOAT):=[[6,0, 1.0, 4.0, -1.0, -1.0]]
--R
--R (7) [6.,0.,1.,4.,- 1.,- 1.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 7

--S 8 of 17
dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]
--R
--R (8) [5.,1.,- 4.,4.,- 4.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 8

--S 9 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
--R
--R (9)
--R [
--R [7.778174591, 0.70710678100000002, - 2.1213203429999998,

```

```

--R      5.6568542480000001, - 3.5355339050000003, - 1.]
--R      ,
--R
--R      [- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
--R      - 2.1213203429999998]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 9

--S 10 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 90 degrees
--R
--R      (10)
--R      [
--R      [4.9999999973618197, 0.99999999947236395, - 3.9999999978894558,
--R      3.9999999978894558, - 3.9999999978894558, - 1.]
--R      ,
--R      [- 5.9999999968341839, 0., - 0.99999999947236429, - 3.9999999978894558,
--R      0.99999999947236429]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 10

--S 11 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 135 degrees
--R
--R      (11)
--R      [
--R      [- 0.70710678062690524, 0.70710678062690502, - 3.535533903134525, 0.,
--R      - 2.1213203418807147, - 1.]
--R      ,
--R      [- 7.7781745868959549, - 0.70710678062690502, 2.1213203418807147,
--R      - 5.6568542450152401, 3.535533903134525]
--R      ]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 11

--S 12 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 180 degrees
--R
--R      (12)
--R      [
--R      [- 5.9999999936683679, 0., - 0.99999999894472813, - 3.9999999957789121,
--R      0.99999999894472813, - 1.]
--R      ,
--R      [- 4.9999999947236393, - 0.99999999894472802, 3.9999999957789116,
--R      - 3.9999999957789121, 3.9999999957789116]

```

```

--R      ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 12

--S 13 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 225 degrees
--R
--R      (13)
--R      [
--R      [- 7.7781745827919098, - 0.70710678025381002, 2.1213203407614296,
--R      - 5.6568542420304802, 3.5355339012690501, - 1.]
--R      ,
--R      [0.70710678025381046, - 0.70710678025381002, 3.5355339012690501, 0.,
--R      2.1213203407614296]
--R      ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 13

--S 14 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 270 degrees
--R
--R      (14)
--R      [
--R      [- 4.999999992085459, - 0.99999999841709197, 3.9999999936683674,
--R      - 3.9999999936683679, 3.9999999936683674, - 1.]
--R      ,
--R      [5.9999999905025518, 0., 0.99999999841709231, 3.9999999936683679,
--R      - 0.99999999841709231]
--R      ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 14

--S 15 of 17
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 315 degrees
--R
--R      (15)
--R      [
--R      [0.70710677988071569, - 0.70710677988071502, 3.5355338994035752, 0.,
--R      2.1213203396421445, - 1.]
--R      ,
--R      [7.7781745786878647, 0.70710677988071502, - 2.1213203396421445,
--R      5.6568542390457202, - 3.5355338994035752]
--R      ]
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 15

--S 16 of 17

```

```

drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate to 360 degrees
--R
--R (16)
--R [
--R [5.9999999873367358, 0., 0.99999999788945637, 3.9999999915578237,
--R - 0.99999999788945637, - 1.]
--R ,
--R [4.9999999894472786, 0.99999999788945593, - 3.9999999915578233,
--R 3.9999999915578237, - 3.9999999915578233]
--R ]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 16

--S 17 of 17
[dx,dy] -- note that the input arguments, dx and dy were modified
--R
--R (17)
--R [
--R [5.9999999873367358, 0., 0.99999999788945637, 3.9999999915578237,
--R - 0.99999999788945637, - 1.]
--R ,
--R [4.9999999894472786, 0.99999999788945593, - 3.9999999915578233,
--R 3.9999999915578237, - 3.9999999915578233]
--R ]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 17

)spool
)lisp (bye)

```

— drot.help —

```

=====
drot examples
=====

```

We set up two arrays of doublefloats, dx and dy

```
dx:PRIMARR(DFLOAT):=[[6,0, 1.0, 4.0, -1.0, -1.0]]
```

```
[6.,0.,1.,4.,- 1.,- 1.]
```

```
dy:PRIMARR(DFLOAT):=[[5.0, 1.0, -4.0, 4.0, -4.0]]
```

```
[5.,1.,- 4.,4.,- 4.]
```

We rotate them by 45 degrees where

```
cos(45) = 0.707106781
sin(45) = 0.707106781
```

```
drot(5,dx,1,dy,1,0.707106781,0.707106781) -- rotate by 45 degrees
```

```
[ [7.778174591, 0.70710678100000002, - 2.1213203429999998,
  5.6568542480000001, - 3.5355339050000003, - 1.],

[- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
- 2.1213203429999998] ]
```

NOTE that the input arguments, dx and dy were modified, per BLAS spec.

```
[dx,dy]
```

```
[ [7.778174591, 0.70710678100000002, - 2.1213203429999998,
  5.6568542480000001, - 3.5355339050000003, - 1.],

[- 0.70710678100000002, 0.70710678100000002, - 3.5355339050000003, 0.,
- 2.1213203429999998] ]
```

We rotate them by -45 degrees where

```
cos(-45) = 0.707106781
sin(-45) = -0.707106781
```

```
drot(5,dx,1,dy,1,0.707106781,-0.707106781) -- rotate by -45 degrees
```

```
[ [5.9999999968341839, 7.8496237287950521E-17, 0.99999999947236384,
  3.9999999978894558, - 0.99999999947236451, - 1.],

[4.9999999973618188, 0.99999999947236384, - 3.9999999978894558,
  3.9999999978894554, - 3.9999999978894554] ]
```

```
=====
Man Page Details
=====
```

NAME

DROT - BLAS level one, plane rotation subroutines

SYNOPSIS

```
SUBROUTINE DROT      ( n, x, incx, y, incy, c, s )
```

```
      INTEGER          n, incx, incy
```

```
      DOUBLE PRECISION x, y, c, s
```

DESCRIPTION

DROT applies a plane rotation matrix to a real sequence of ordered pairs:

$$(x_i, y_i), \text{ for all } i = 1, 2, \dots, n.$$

ARGUMENTS

- n** INTEGER. (input)
Number of ordered pairs (planar points in DROT) to be rotated. If $n \leq 0$, this routine returns without computation.
- x** DOUBLE PRECISION, (input and output)
Array of dimension $(n-1) * |incx| + 1$. On input, array x contains the x-coordinate of each planar point to be rotated. On output, array x contains the x-coordinate of each rotated planar point.
- incx** INTEGER. (input)
Increment between elements of x. If $incx = 0$, the results will be unpredictable.
- y** DOUBLE PRECISION, (input and output)
array of dimension $(n-1) * |incy| + 1$.
On input, array y contains the y-coordinate of each planar point to be rotated. On output, array y contains the y-coordinate of each rotated planar point.
- incy** INTEGER. (input)
Increment between elements of y. If $incy = 0$, the results will be unpredictable.
- c** DOUBLE PRECISION, Cosine of the angle of rotation. (input)
- s** DOUBLE PRECISION, Sine of the angle of rotation. (input)

NOTES

This routine applies the following plane rotation to each pair of elements (x_i, y_i) :

$$\begin{bmatrix} x(i) \\ y(i) \end{bmatrix} \leftarrow \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} x(i) \\ y(i) \end{bmatrix}$$

for $i = 1, \dots, n$

If coefficients c and s satisfy $c^2 + s^2 = 1.0$, the rotation matrix

is orthogonal, and the transformation is called a Givens plane rotation. If $c = 1$ and $s = 0$, DROT returns without modifying any input parameters.

To calculate the Givens coefficients c and s from a two-element vector to determine the angle of rotation, use SROTG(3S).

When working backward ($incx < 0$ or $incy < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

$y(1-incy * (n-1)), y(1-incy * (n-2)), \dots, y(1)$

—————

— drot.f —

```

subroutine drot (n,dx,incx,dy,incy,c,s)
c
c  applies a plane rotation.
c  jack dongarra, linpack, 3/11/78.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
c  double precision dx(*),dy(*),dtemp,c,s
c  integer i,incx,incy,ix,iy,n
c
c  if(n.le.0)return
c  if(incx.eq.1.and.incy.eq.1)go to 20
c
c      code for unequal increments or equal increments not equal
c      to 1
c
c  ix = 1
c  iy = 1
c  if(incx.lt.0)ix = (-n+1)*incx + 1
c  if(incy.lt.0)iy = (-n+1)*incy + 1
c  do 10 i = 1,n
c      dtemp = c*dx(ix) + s*dy(iy)
c      dy(iy) = c*dy(iy) - s*dx(ix)
c      dx(ix) = dtemp
c      ix = ix + incx
c      iy = iy + incy
10 continue
c  return
c
c      code for both increments equal to 1
c

```

```

20 do 30 i = 1,n
    dtemp = c*dx(i) + s*dy(i)
    dy(i) = c*dy(i) - s*dx(i)
    dx(i) = dtemp
30 continue
return
end

```

— drotg example —

```

program drotEX
*   Tim Daly May 4, 2012
*   unit tests for BLAS drot
double precision dx(5),dy(5)
double precision c,s

write(6,1)
1  format(/,"rotate by 45, -45")
   dx= (/ 6.0d0, 1.0d0, 4.0d0, 1.0d0, -1.0d0 /)
   dy= (/ 5.0d0, 1.0d0, -4.0d0, 4.0d0, -4.0d0 /)
   write(6,10)dx(1),dy(1)
10  format(/,"dx(1)=",f6.3," dy(1)=",f6.3)
   write(6,20)dx(2),dy(2)
20  format("dx(2)=",f6.3," dy(2)=",f6.3)
   write(6,30)dx(3),dy(3)
30  format("dx(3)=",f6.3," dy(2)=",f6.3)
   write(6,40)dx(4),dy(4)
40  format("dx(4)=",f6.3," dy(4)=",f6.3)
   write(6,50)dx(5),dy(5)
50  format("dx(5)=",f6.3," dy(5)=",f6.3)
   call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
   write(6,10)dx(1),dy(1)
   write(6,20)dx(2),dy(2)
   write(6,30)dx(3),dy(3)
   write(6,40)dx(4),dy(4)
   write(6,50)dx(5),dy(5)
   call drot(5,dx,1,dy,1,0.707106781d0,-0.707106781d0)
   write(6,10)dx(1),dy(1)
   write(6,20)dx(2),dy(2)
   write(6,30)dx(3),dy(3)
   write(6,40)dx(4),dy(4)
   write(6,50)dx(5),dy(5)

write(6,2)
2  format(/,"rotate by 45, -45, only some members")
   dx= (/ 6.0d0, 1.0d0, 4.0d0, 1.0d0, -1.0d0 /)

```

```

dy= (/ 5.0d0, 1.0d0, -4.0d0, 4.0d0, -4.0d0 /)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(3,dx,2,dy,2,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(3,dx,2,dy,2,0.707106781d0,-0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)

write(6,3)
3 format(/,"rotate by 360")
dx= (/ 6.0d0, 1.0d0, 4.0d0, 1.0d0, -1.0d0 /)
dy= (/ 5.0d0, 1.0d0, -4.0d0, 4.0d0, -4.0d0 /)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)

```

```

write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)
call drot(5,dx,1,dy,1,0.707106781d0,0.707106781d0)
write(6,10)dx(1),dy(1)
write(6,20)dx(2),dy(2)
write(6,30)dx(3),dy(3)
write(6,40)dx(4),dy(4)
write(6,50)dx(5),dy(5)

4  write(6,4)
   format(/,"rotate by 45, from end")
   dx= (/ 6.0d0, 1.0d0, 4.0d0, 1.0d0, -1.0d0 /)
   dy= (/ 5.0d0, 1.0d0, -4.0d0, 4.0d0, -4.0d0 /)
   write(6,10)dx(1),dy(1)
   write(6,20)dx(2),dy(2)
   write(6,30)dx(3),dy(3)
   write(6,40)dx(4),dy(4)
   write(6,50)dx(5),dy(5)
   call drot(2,dx,-1,dy,-1,0.707106781d0,0.707106781d0)
   write(6,10)dx(1),dy(1)
   write(6,20)dx(2),dy(2)
   write(6,30)dx(3),dy(3)
   write(6,40)dx(4),dy(4)
   write(6,50)dx(5),dy(5)
   stop
   end

```

```
gcc -o drotEX drotEX.f -lgfortran drot.o && ./drotEX
```

```
rotate by 45, -45
```

```
dx(1)= 6.000 dy(1)= 5.000
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

```
dx(1)= 7.778 dy(1)=-0.707
dx(2)= 1.414 dy(2)= 0.000
dx(3)= 0.000 dy(2)=-5.657
dx(4)= 2.121 dy(4)= 3.536
dx(5)=-3.536 dy(5)=-2.121
```

```
dx(1)= 6.000 dy(1)= 5.000
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

rotate by 45, -45, only some members

```
dx(1)= 6.000 dy(1)= 5.000
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

```
dx(1)= 7.778 dy(1)=-0.707
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 0.000 dy(2)=-5.657
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-3.536 dy(5)=-2.121
```

```
dx(1)= 6.000 dy(1)= 5.000
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

rotate by 360

```
dx(1)= 6.000 dy(1)= 5.000
dx(2)= 1.000 dy(2)= 1.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

```
dx(1)= 7.778 dy(1)=-0.707
dx(2)= 1.414 dy(2)= 0.000
dx(3)= 0.000 dy(2)=-5.657
```

dx(4)= 2.121 dy(4)= 3.536
 dx(5)=-3.536 dy(5)=-2.121

dx(1)= 5.000 dy(1)=-6.000
 dx(2)= 1.000 dy(2)=-1.000
 dx(3)=-4.000 dy(2)=-4.000
 dx(4)= 4.000 dy(4)= 1.000
 dx(5)=-4.000 dy(5)= 1.000

dx(1)=-0.707 dy(1)=-7.778
 dx(2)= 0.000 dy(2)=-1.414
 dx(3)=-5.657 dy(2)= 0.000
 dx(4)= 3.536 dy(4)=-2.121
 dx(5)=-2.121 dy(5)= 3.536

dx(1)=-6.000 dy(1)=-5.000
 dx(2)=-1.000 dy(2)=-1.000
 dx(3)=-4.000 dy(2)= 4.000
 dx(4)= 1.000 dy(4)=-4.000
 dx(5)= 1.000 dy(5)= 4.000

dx(1)=-7.778 dy(1)= 0.707
 dx(2)=-1.414 dy(2)= 0.000
 dx(3)= 0.000 dy(2)= 5.657
 dx(4)=-2.121 dy(4)=-3.536
 dx(5)= 3.536 dy(5)= 2.121

dx(1)=-5.000 dy(1)= 6.000
 dx(2)=-1.000 dy(2)= 1.000
 dx(3)= 4.000 dy(2)= 4.000
 dx(4)=-4.000 dy(4)=-1.000
 dx(5)= 4.000 dy(5)=-1.000

dx(1)= 0.707 dy(1)= 7.778
 dx(2)= 0.000 dy(2)= 1.414
 dx(3)= 5.657 dy(2)= 0.000
 dx(4)=-3.536 dy(4)= 2.121
 dx(5)= 2.121 dy(5)=-3.536

dx(1)= 6.000 dy(1)= 5.000
 dx(2)= 1.000 dy(2)= 1.000
 dx(3)= 4.000 dy(2)=-4.000
 dx(4)=-1.000 dy(4)= 4.000
 dx(5)=-1.000 dy(5)=-4.000

rotate by 45, from end

dx(1)= 6.000 dy(1)= 5.000
 dx(2)= 1.000 dy(2)= 1.000
 dx(3)= 4.000 dy(2)=-4.000

```
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

```
dx(1)= 7.778 dy(1)=-0.707
dx(2)= 1.414 dy(2)= 0.000
dx(3)= 4.000 dy(2)=-4.000
dx(4)=-1.000 dy(4)= 4.000
dx(5)=-1.000 dy(5)=-4.000
```

— BLAS 1 drot —

```
(defun drot (n dx incx dy incy c s)
; Tim Daly May 4, 2012
  (declare (type (double-float) c s)
            (type (simple-array double-float (*)) dx dy)
            (type fixnum incy incx n))
  (let ((dtemp 0.0d0) (ix 0) (iy 0))
    (declare (type (double-float) dtemp) (type fixnum iy ix))
    (when (> n 0)
      (if (< incx 0)
        (setf ix (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incx))))
      (if (< incy 0)
        (setf iy (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incy))))
      (dotimes (i n)
        (setf dtemp (the double-float
          (+ (the double-float (* c (the double-float (svref dx ix))))
            (the double-float (* s (the double-float (svref dy iy)))))))
        (setf (the double-float (svref dy iy)) (the double-float
          (- (* c (the double-float (svref dy iy)))
            (* s (the double-float (svref dx ix))))))
        (setf (the double-float (svref dx ix)) (the double-float dtemp))
        (setf ix (the fixnum (+ ix incx)))
        (setf iy (the fixnum (+ iy incy))))))
    (list dx dy)))
```

— BLAS 1 drot lisp test —

```
(load "drot.lisp")
; rotate by 45, -45
(setq dx (vector 6.0d0 1.0d0 4.0d0 -1.0d0 -1.0d0))
; #(6.0 1.0 4.0 -1.0 -1.0)

(setq dy (vector 5.0d0 1.0d0 -4.0d0 4.0d0 -4.0d0))
```

```

; #(5.0 1.0 -4.0 4.0 -4.0)

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
; (#(7.778174591 1.414213562 0.0 2.1213203429999998 -3.5355339050000003)
;   #(-0.70710678099999935 0.0 -5.6568542480000001 3.5355339050000003
;     -2.1213203429999998))

(drot 5 dx 1 dy 1 0.707106781d0 -0.707106781d0)
; (#(5.9999999968341831 0.99999999947236395 3.9999999978894558
;   -0.99999999947236451 -0.99999999947236451)
;   #(-4.9999999973618197 0.99999999947236395 -3.9999999978894558
;     3.9999999978894563 -3.9999999978894563))

; rotate by 45, -45, only some members

(setq dx (vector 6.0d0 1.0d0 4.0d0 -1.0d0 -1.0d0))
; #(6.0 1.0 4.0 -1.0 -1.0)

(setq dy (vector 5.0d0 1.0d0 -4.0d0 4.0d0 -4.0d0))
; #(5.0 1.0 -4.0 4.0 -4.0)

(drot 3 dx 2 dy 2 0.707106781d0 0.707106781d0)
; (#(7.778174591 1.0 0.0 -1.0 -3.5355339050000003)
;   #(-0.70710678099999935 1.0 -5.6568542480000001 4.0 -2.1213203429999998))

(drot 3 dx 2 dy 2 0.707106781d0 -0.707106781d0)
; (#(5.9999999968341831 1.0 3.9999999978894558 -1.0 -0.99999999947236451)
;   #(-4.9999999973618197 1.0 -3.9999999978894558 4.0 -3.9999999978894563))

; rotate by 360

(setq dx (vector 6.0d0 1.0d0 4.0d0 -1.0d0 -1.0d0))
; #(6.0 1.0 4.0 -1.0 -1.0)

(setq dy (vector 5.0d0 1.0d0 -4.0d0 4.0d0 -4.0d0))
; #(5.0 1.0 -4.0 4.0 -4.0)

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
; (#(7.778174591 1.414213562 0.0 2.1213203429999998 -3.5355339050000003)
;   #(-0.70710678099999935 0.0 -5.6568542480000001 3.5355339050000003
;     -2.1213203429999998))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
; (#(4.9999999973618197 0.99999999947236395 -3.9999999978894558
;   3.9999999978894563 -3.9999999978894563)
;   #(-5.9999999968341831 -0.99999999947236395 -3.9999999978894558
;     0.99999999947236451 0.99999999947236451))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
; (#(-0.70710678062690446 0.0 -5.6568542450152401 3.5355339031345254

```



```

;      -2.1213203418807147)
;      #(-7.778174586895954 -1.41421356125381 0.0 -2.1213203418807147
;      3.5355339031345254))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
;      (#(-5.9999999936683661 -0.99999999894472802 -3.9999999957789121
;      0.99999999894472835 0.99999999894472835)
;      #(-4.9999999947236393 -0.99999999894472802 3.9999999957789121
;      -3.9999999957789116 3.9999999957789116))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
;      (#(-7.7781745827919089 -1.41421356050762 0.0 -2.1213203407614296
;      3.5355339012690505)
;      #(-0.70710678025380957 0.0 5.6568542420304802 -3.5355339012690505
;      2.1213203407614296))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
;      (#(-4.999999992085459 -0.99999999841709197 3.9999999936683679
;      -3.9999999936683679 3.9999999936683679)
;      #(-5.999999990502551 0.99999999841709197 3.9999999936683679
;      -0.99999999841709286 -0.99999999841709286))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
;      (#(0.70710677988071557 0.0 5.6568542390457202 -3.5355338994035757
;      2.1213203396421445)
;      #(7.7781745786878638 1.41421355976143 0.0 2.1213203396421445
;      -3.5355338994035757))

(drot 5 dx 1 dy 1 0.707106781d0 0.707106781d0)
;      (#(5.9999999873367349 0.99999999788945593 3.9999999915578237
;      -0.9999999978894567 -0.9999999978894567)
;      #(4.9999999894472777 0.99999999788945593 -3.9999999915578237
;      3.9999999915578237 -3.9999999915578237))

; rotate by 45, -45 from end

(setq dx (vector 6.0d0 1.0d0 4.0d0 -1.0d0 -1.0d0))
;      #(6.0 1.0 4.0 -1.0 -1.0)

(setq dy (vector 5.0d0 1.0d0 -4.0d0 4.0d0 -4.0d0))
;      #(5.0 1.0 -4.0 4.0 -4.0)

(drot 2 dx -1 dy -1 0.707106781d0 0.707106781d0)
;      (#(7.778174591 1.414213562 4.0 -1.0 -1.0)
;      #(-0.70710678099999935 0.0 -4.0 4.0 -4.0))

```

dscal BLAS

— dscal.input —

```

)set break resume
)sys rm -f dscal.output
)spool dscal.output
)set message test on
)set message auto off
)clear all

--S 1 of 6
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
--R
--R (1) [1.,2.,3.,4.,5.,6.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 6
dscal(6,2.0,dx,1)
--R
--R (2) [2.,4.,6.,8.,10.,12.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 6
dx
--R
--R (3) [2.,4.,6.,8.,10.,12.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 3

--S 4 of 6
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
--R
--R (4) [1.,2.,3.,4.,5.,6.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 4

--S 5 of 6
dscal(3,0.5,dx,1)
--R
--R (5) [0.5,1.,1.5,4.,5.,6.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 5

--S 6 of 6
dx
```

```
--R
--R  (6)  [0.5,1.,1.5,4.,5.,6.]
--R                                          Type: PrIMITIVEArray(DoubleFloat)
--E 6
```

```
)spool
)lisp (bye)
```

— dscal.help —

```
=====
dscal examples
=====
```

Given an initial array

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
[1.,2.,3.,4.,5.,6.]
```

scale each element by 2.0

```
dscal(6,2.0,dx,1)
```

```
[2.,4.,6.,8.,10.,12.]
```

Note that this changes the initial array in place.

```
dx
```

```
[2.,4.,6.,8.,10.,12.]
```

Given the initial array

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
[1.,2.,3.,4.,5.,6.]
```

scale each element by 0.5

```
dscal(6,0.5,dx,1)
```

```
[0.5,1.,1.5,2.,2.5,3.]
```

Given the initial array

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
[1.,2.,3.,4.,5.,6.]
```

scale the first three elements by 0.5

```
dscal(3,0.5,dx,1)
```

```
[0.5,1.,1.5,4.,5.,6.]
```

Given the initial array

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
[1.,2.,3.,4.,5.,6.]
```

dscal will do nothing with a negative count

```
dscal(-3,0.5,dx,1)
```

```
[1.,2.,3.,4.,5.,6.]
```

Given the initial array

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]
```

```
[1.,2.,3.,4.,5.,6.]
```

dscal will do nothing with a negative increment

```
dscal(3,0.5,dx,-1)
```

```
[1.,2.,3.,4.,5.,6.]
```

```
=====
Man Page Details
=====
```

NAME

DSCAL - BLAS level one, scales a double precision vector

SYNOPSIS

```
SUBROUTINE DSCAL      ( n, alpha, x, incx )
```

```
INTEGER              n, incx
```

```
DOUBLE PRECISION    alpha, x
```

DESCRIPTION

DSCAL scales a double precision vector with a double precision scalar. DSCAL scales the vector x of length n and increment incx by the constant a.

This routine performs the following vector operation:

$$x \leftarrow \alpha x$$

where alpha is a double precision scalar, and x is a double precision vector.

ARGUMENTS

n INTEGER. (input)
 Number of elements in the vector.
 If n <= 0, this routine returns without computation.

alpha DOUBLE PRECISION scalar alpha. (input)

x DOUBLE PRECISION, (input and output)
 Array of dimension (n-1) * |incx| + 1. Vector to be scaled.

incx INTEGER. (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

— dscal.f —

```

subroutine dscal(n,da,dx,incx)
c
c  scales a vector by a constant.
c  uses unrolled loops for increment equal to one.
c  jack dongarra, linpack, 3/11/78.
c  modified 3/93 to return if incx .le. 0.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
  double precision da,dx(*)
  integer i,incx,m,mp1,n,nincx
c
  if( n.le.0 .or. incx.le.0 )return
  if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
  nincx = n*incx
  do 10 i = 1,nincx,incx

```

```

        dx(i) = da*dx(i)
10 continue
    return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
c
20 m = mod(n,5)
    if( m .eq. 0 ) go to 40
    do 30 i = 1,m
        dx(i) = da*dx(i)
30 continue
    if( n .lt. 5 ) return
40 mp1 = m + 1
    do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
50 continue
    return
end

```

— drotg example —

```

program dscalEX
*   Tim Daly May 6, 2012
*   unit tests for BLAS dscal
double precision a(6)
double precision b
a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)

write(6,10)
10  format(/,"the initial array")
    write(6,100)a(1),a(2),a(3)
100 format("a(1)=",f6.3," a(2)=",f6.3," a(3)=",f6.3)
    write(6,200)a(4),a(5),a(6)
200 format("a(4)=",f6.3," a(5)=",f6.3," a(6)=",f6.3)

write(6,11)
11  format(/,"scale whole array by 2.0")
    call dscal(6,2.0d0,a,1)
    write(6,100)a(1),a(2),a(3)

```

```

        write(6,200)a(4),a(5),a(6)

        write(6,12)
12      format(/,"scale whole array by 0.5")
        a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)
        call dscal(6,0.5d0,a,1)
        write(6,100)a(1),a(2),a(3)
        write(6,200)a(4),a(5),a(6)

        write(6,13)
13      format(/,"scale 3 elements by 0.5")
        a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)
        call dscal(3,0.5d0,a,1)
        write(6,100)a(1),a(2),a(3)
        write(6,200)a(4),a(5),a(6)

        write(6,14)
14      format(/,"do nothing. negative length")
        a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)
        call dscal(-3,0.5d0,a,1)
        write(6,100)a(1),a(2),a(3)
        write(6,200)a(4),a(5),a(6)

        write(6,15)
15      format(/,"scale 3 elements by -0.5, from end (does nothing)")
        a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0 /)
        call dscal(3,0.5d0,a,-1)
        write(6,100)a(1),a(2),a(3)
        write(6,200)a(4),a(5),a(6)

        stop
        end

```

```
gcc -o dscalEX dscalEX.f -lgfortran dscal.o && ./dscalEX
```

```

the initial array
a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000

scale whole array by 2.0
a(1)= 2.000 a(2)= 4.000 a(3)= 6.000
a(4)= 8.000 a(5)=10.000 a(6)=12.000

scale whole array by 0.5
a(1)= 0.500 a(2)= 1.000 a(3)= 1.500
a(4)= 2.000 a(5)= 2.500 a(6)= 3.000

```

```

scale 3 elements by 0.5
a(1)= 0.500 a(2)= 1.000 a(3)= 1.500
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000

do nothing. negative length
a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000

scale 3 elements by -0.5, from end (does nothing)
a(1)= 1.000 a(2)= 2.000 a(3)= 3.000
a(4)= 4.000 a(5)= 5.000 a(6)= 6.000

```

— BLAS 1 dscal —

```

(defun dscal (n da dx incx)
; Tim Daly May 6, 2012
  (declare (type (simple-array double-float (*)) dx)
            (type (double-float) da)
            (type fixnum n incx))
  (let ((i 0) (limit (length dx)))
    (declare (type fixnum limit i))
    (when (and (> n 0) (> incx 0))
      (do ((i 0 (the fixnum (+ i incx))))
          ((or (>= i n) (>= i limit)))
        (setf (svref dx i)
              (the double-float (* da (the double-float (svref dx i))))))
      dx))

```

— BLAS 1 dscal lisp test —

```

(load "dscal.lisp")
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0))
; #(1.0 2.0 3.0 4.0 5.0 6.0)
(dscal 6 2.0d0 a 1)
; #(2.0 4.0 6.0 8.0 10.0 12.0)

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0))
; #(1.0 2.0 3.0 4.0 5.0 6.0)
(dscal 6 0.5d0 a 1)
; #(0.5 1.0 1.5 2.0 2.5 3.0)

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0))
; #(1.0 2.0 3.0 4.0 5.0 6.0)
(dscal 3 0.5d0 a 1)

```



```
; #(0.5 1.0 1.5 4.0 5.0 6.0)

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0))
; #(1.0 2.0 3.0 4.0 5.0 6.0)
(dscal -3 0.5d0 a 1)
; #(1.0 2.0 3.0 4.0 5.0 6.0)

(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0 6.0d0))
; #(1.0 2.0 3.0 4.0 5.0 6.0)
(dscal 3 0.5d0 a -1)
; #(1.0 2.0 3.0 4.0 5.0 6.0)
```

dswap BLAS

— dswap.input —

```
)set break resume
)sys rm -f dswap.output
)spool dswap.output
)set message test on
)set message auto off
)clear all

--S 1 of 9
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R (1) [1.,2.,3.,4.,5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 9
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R (2) [9.,8.,7.,6.,- 5.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 2

--S 3 of 9
dswap(5,dx,1,dy,1)
--R
--R (3) [[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
--R
--R                                          Type: List(PrimitiveArray(DoubleFloat))
--E 3
```

```

--S 4 of 9
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R (4) [1.,2.,3.,4.,5.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 4

--S 5 of 9
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R (5) [9.,8.,7.,6.,- 5.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 5

--S 6 of 9
dswap(3,dx,2,dy,2)
--R
--R (6) [[9.,2.,7.,4.,- 5.],[1.,8.,3.,6.,5.]]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 6

--S 7 of 9
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
--R
--R (7) [1.,2.,3.,4.,5.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 7

--S 8 of 9
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
--R
--R (8) [9.,8.,7.,6.,- 5.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 8

--S 9 of 9
dswap(5,dx,1,dy,-1)
--R
--R (9) [[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
--R
--R                                         Type: List(PrimitiveArray(DoubleFloat))
--E 9

)spool
)lisp (bye)

```

```
=====
dswap examples
=====
```

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
```

```
[1.,2.,3.,4.,5.]
```

```
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
```

```
[9.,8.,7.,6.,- 5.]
```

```
dswap(5,dx,1,dy,1)
```

```
[[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
```

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
```

```
[1.,2.,3.,4.,5.]
```

```
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
```

```
[9.,8.,7.,6.,- 5.]
```

```
dswap(3,dx,2,dy,2)
```

```
[[9.,2.,7.,4.,- 5.],[1.,8.,3.,6.,5.]]
```

```
dx:PRIMARR(DFLOAT):=[[1.0, 2.0, 3.0, 4.0, 5.0]]
```

```
[1.,2.,3.,4.,5.]
```

```
dy:PRIMARR(DFLOAT):=[[9.0, 8.0, 7.0, 6.0, -5.0]]
```

```
[9.,8.,7.,6.,- 5.]
```

```
dswap(5,dx,1,dy,-1)
```

```
[[9.,8.,7.,6.,- 5.],[1.,2.,3.,4.,5.]]
```

```
=====
Man Page Details
=====
```

NAME

DSWAP - BLAS level one, Swaps two double precision vectors

SYNOPSIS

```
SUBROUTINE DSWAP    ( n, x, incx, y, incy )
```

INTEGER n, incx, incy

DOUBLE PRECISION x, y

DESCRIPTION

DSWAP swaps two double precision vectors, it interchanges n values of vector x and vector y . $incx$ and $incy$ specify the increment between two consecutive elements of respectively vector x and y .

This routine performs the following vector operation:

$$x \leftrightarrow y$$

where x and y are double precision vectors.

ARGUMENTS

n INTEGER. (input)
 Number of vector elements to be swapped.
 If $n \leq 0$, this routine returns without computation.

x DOUBLE PRECISION, (input and output)
 Array of dimension $(n-1) * |incx| + 1$.

$incx$ INTEGER. (input)
 Increment between elements of x .
 If $incx = 0$, the results will be unpredictable.

y DOUBLE PRECISION, (input and output)
 array of dimension $(n-1) * |incy| + 1$. Vector to be swapped.

$incy$ INTEGER. (input)
 Increment between elements of y . If $incy = 0$, the results will be unpredictable.

NOTES

When working backward ($incx < 0$ or $incy < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

$y(1-incy * (n-1)), y(1-incy * (n-2)), \dots, y(1)$

— dswap.f —

```

      subroutine dswap (n,dx,incx,dy,incy)
c
c   interchanges two vectors.
c   uses unrolled loops for increments equal one.
c   jack dongarra, linpack, 3/11/78.
c   modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments not equal
c       to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
         dtemp = dx(ix)
         dx(ix) = dy(iy)
         dy(iy) = dtemp
         ix = ix + incx
         iy = iy + incy
10 continue
      return
c
c       code for both increments equal to 1
c
c       clean-up loop
c
20 m = mod(n,3)
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
      dtemp = dx(i)
      dx(i) = dy(i)
      dy(i) = dtemp
30 continue
   if( n .lt. 3 ) return
40 mp1 = m + 1
   do 50 i = mp1,n,3
      dtemp = dx(i)
      dx(i) = dy(i)
      dy(i) = dtemp
      dtemp = dx(i + 1)
      dx(i + 1) = dy(i + 1)
      dy(i + 1) = dtemp

```

```

        dtemp = dx(i + 2)
        dx(i + 2) = dy(i + 2)
        dy(i + 2) = dtemp
50 continue
    return
end

```

There appears to be a bit of confusion using negative increments. The fortran code does not work as I would have guessed. I don't understand what it was intended to do.

— dswap example —

```

program dswapEX
*   Tim Daly May 6, 2012
*   unit tests for BLAS dswap
double precision a(5)
double precision b(5)
a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0/)
b = (/ 9.0D0, 8.0D0, 7.0D0, 6.0D0, -5.0D0/)

write(6,10)
10  format(/,"before:")
    write(6,100)a(1),a(2),a(3),a(4),a(5)
100 format("dx=(",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3,")")
    write(6,101)b(1),b(2),b(3),b(4),b(5)
101 format("dy=(",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3,")")
    write(6,11)
11  format(/,"swap all elements from dx to dy");
    call dswap(5,a,1,b,1)
    write(6,100)a(1),a(2),a(3),a(4),a(5)
    write(6,101)b(1),b(2),b(3),b(4),b(5)

write(6,10)
a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0/)
b = (/ 9.0D0, 8.0D0, 7.0D0, 6.0D0, -5.0D0/)
write(6,100)a(1),a(2),a(3),a(4),a(5)
write(6,101)b(1),b(2),b(3),b(4),b(5)
write(6,20)
20  format(/,"swap every other element from dx to dy");
    a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0/)
    b = (/ 9.0D0, 8.0D0, 7.0D0, 6.0D0, -5.0D0/)
    call dswap(3,a,2,b,2)
    write(6,100)a(1),a(2),a(3),a(4),a(5)
    write(6,101)b(1),b(2),b(3),b(4),b(5)

write(6,10)

```

```

      a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0/)
      b = (/ 9.0D0, 8.0D0, 7.0D0, 6.0D0, -5.0D0/)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
      write(6,101)b(1),b(2),b(3),b(4),b(5)
      write(6,22)
22    format(/,"swap and reverse elements dx to dy");
      a = (/ 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0/)
      b = (/ 9.0D0, 8.0D0, 7.0D0, 6.0D0, -5.0D0/)
      call dswap(5,a,1,b,-1)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      stop
      end

```

```
gcc -o dswapEX dswapEX.f -lgfortran dswap.o && ./dswapEX
```

before:

```
dx=(/ 1.000, 2.000, 3.000, 4.000, 5.000/)
dy=(/ 9.000, 8.000, 7.000, 6.000, -5.000/)
```

swap all elements from dx to dy

```
dx=(/ 9.000, 8.000, 7.000, 6.000, -5.000/)
dy=(/ 1.000, 2.000, 3.000, 4.000, 5.000/)
```

before:

```
dx=(/ 1.000, 2.000, 3.000, 4.000, 5.000/)
dy=(/ 9.000, 8.000, 7.000, 6.000, -5.000/)
```

swap every other element from dx to dy

```
dx=(/ 9.000, 2.000, 7.000, 4.000, -5.000/)
dy=(/ 1.000, 8.000, 3.000, 6.000, 5.000/)
```

before:

```
dx=(/ 1.000, 2.000, 3.000, 4.000, 5.000/)
dy=(/ 9.000, 8.000, 7.000, 6.000, -5.000/)
```

swap and reverse elements dx to dy

```
dx=(/-5.000, 6.000, 7.000, 8.000, 9.000/)
dy=(/ 5.000, 4.000, 3.000, 2.000, 1.000/)
```

— BLAS 1 dswap —

```
(defun dswap (n dx incx dy incy)
; Tim Daly May 6, 2012
```

```

(declare (type (simple-array double-float (*)) dy dx)
  (type fixnum incy incx n))
(let ((ix 0) (iy 0) (limitx (length dx)) (limity (length dy))
      (dtemp 0.0d0))
(declare (type (double-float) dtemp) (type fixnum ix iy limitx limity))
(when (> n 0)
  (when (< incx 0)
    (setf ix (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incx))))
  (when (< incy 0)
    (setf iy (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incy))))
  (do ((i 0 (1+ i)))
      ((or (>= i n) (>= i limitx) (>= i limity)))
    (setf dtemp (svref dx ix))
    (setf (svref dx ix) (svref dy iy))
    (setf (svref dy iy) dtemp)
    (setf ix (the fixnum (+ ix incx)))
    (setf iy (the fixnum (+ iy incy)))))
(list dx dy)))

```

— BLAS 1 dswap lisp test —

```

(load "dswap.lisp")
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
; #(1.0 2.0 3.0 4.0 5.0)
(setq b (vector 9.0d0 8.0d0 7.0d0 6.0d0 -5.0d0))
; #(9.0 8.0 7.0 6.0 -5.0)
(dswap 5 a 1 b 1)
; (#(9.0 8.0 7.0 6.0 -5.0) #(1.0 2.0 3.0 4.0 5.0))
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
; #(1.0 2.0 3.0 4.0 5.0)
(setq b (vector 9.0d0 8.0d0 7.0d0 6.0d0 -5.0d0))
; #(9.0 8.0 7.0 6.0 -5.0)
(dswap 3 a 2 b 2)
; (#(9.0 2.0 7.0 4.0 -5.0) #(1.0 8.0 3.0 6.0 5.0))
(setq a (vector 1.0d0 2.0d0 3.0d0 4.0d0 5.0d0))
; #(1.0 2.0 3.0 4.0 5.0)
(setq b (vector 9.0d0 8.0d0 7.0d0 6.0d0 -5.0d0))
; #(9.0 8.0 7.0 6.0 -5.0)
(dswap 5 a 1 b -1)
; (#(-5.0 6.0 7.0 8.0 9.0) #(5.0 4.0 3.0 2.0 1.0))

```

dzasum BLAS

— dzasum.input —

```
)set break resume
)sys rm -f dzasum.output
)spool dzasum.output
)set message test on
)set message auto off
)clear all

--S 1 of 4
d:PRIMARR(COMPLEX(DFLOAT)):= [[1.0+2.0*i,-3.0+4.0*i,5.0-6.0*i]]
--R
--R (1) [1. + 2. %i,- 3. + 4. %i,5. - 6. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 1

--S 2 of 4
dzasum(3,d,1) -- 21.0
--R
--R (2) 21.
--R
--R                                         Type: DoubleFloat
--E 2

--S 3 of 4
dzasum(3,d,2) -- 14.0
--R
--R (3) 14.
--R
--R                                         Type: DoubleFloat
--E 3

--S 4 of 4
dzasum(-3,d,1) -- 0.0
--R
--R (4) 0.
--R
--R                                         Type: DoubleFloat
--E 4

)spool
)lisp (bye)
```

— dzasum.help —

=====

dzasum examples

```
=====
d:PRIMARR(COMPLEX(DFLOAT)):=[[1.0+2.0*i,-3.0+4.0*i,5.0-6.0*i]]

[1. + 2. %i,- 3. + 4. %i,5. - 6. %i]

dzasum(3,d,1)

21. = (1+2) + (abs(-3)+4) + (5+abs(-6))

dzasum(3,d,2)

14. = (1+2) + (5+abs(-6))

dzasum(-3,d,1)

0.  -- never execute for negative arguments
=====
```

Man Page Details

NAME

DZASUM - BLAS level one, sums the absolute values of the real and imaginary parts of the elements of a double complex vector

SYNOPSIS

DOUBLE PRECISION FUNCTION DZASUM (n, x, incx)

INTEGER n, incx

DOUBLE COMPLEX x

DESCRIPTION

This routine performs the following vector operation:

$$\text{DZASUM} \leftarrow \sum_{i=1}^n \text{abs}(\text{real}(x(i))) + \text{abs}(\text{aimag}(x(i)))$$

ARGUMENTS

n INTEGER. (input)
Number of vector elements to be summed.

x DOUBLE COMPLEX. (input)
Array of dimension (n-1) * abs(incx) + 1.
Vector that contains elements to be summed.

incx INTEGER. (input)

Increment between elements of x.
 If incx = 0, the results will be unpredictable.

RETURN VALUES

DZASUM DOUBLE PRECISION. (output)
 Sum of the absolute values of the real and imaginary parts of
 the elements of the vector x.
 If n <= 0, DZASUM is set to 0.

NOTES

When working backward (incx < 0), each routine starts at the end of the
 vector and moves backward, as follows:

$x(1 - \text{incx} * (n - 1)), x(1 - \text{incx} * (n - 2)), \dots, x(1)$

Computes (complex double-float) $asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$

— dzasum.f —

```

double precision function dzasum(n,zx,incx)
c
c   takes the sum of the absolute values.
c   jack dongarra, 3/11/78.
c   modified 3/93 to return if incx .le. 0.
c   modified 12/3/93, array(1) declarations changed to array(*)
c
  double complex zx(*)
  double precision stemp,dcabs1
  integer i,incx,ix,n
c
  dzasum = 0.0d0
  stemp = 0.0d0
  if( n.le.0 .or. incx.le.0 )return
  if(incx.eq.1)go to 20
c
c       code for increment not equal to 1
c
  ix = 1
  do 10 i = 1,n
    stemp = stemp + dcabs1(zx(ix))
    ix = ix + incx
  10 continue
  dzasum = stemp
  return
c
c       code for increment equal to 1

```

```

c
20 do 30 i = 1,n
    stemp = stemp + dcabs1(zx(i))
30 continue
dzasum = stemp
return
end

```

— dzasum example —

```

program dzasumEX
*   Tim Daly May 7, 2012
*   unit tests for BLAS dzasum
double complex a(3)
double precision b
a = (/ (1.0D0,2.0D0), (-3.0D0,4.0D0), (5.0D0,-6.0D0)/)

write(6,10)
10  format(/,"before:")
write(6,100)dbble(a(1)),dimag(a(1))
100 format("a(1)=C(",f6.3," ",f6.3,")")
write(6,101)dbble(a(2)),dimag(a(2))
101 format("a(2)=C(",f6.3," ",f6.3,")")
write(6,102)dbble(a(3)),dimag(a(3))
102 format("a(3)=C(",f6.3," ",f6.3,")")

b=dzasum(3,a,1)
write(6,11)
11  format(/,"should be 21:")
write(6,200)b
200 format("b=",f6.3)

b=dzasum(3,a,2)
write(6,12)
12  format(/,"should be 14:")
write(6,200)b

b=dzasum(-3,a,1)
write(6,13)
13  format(/,"should be 0:")
write(6,200)b

stop
end

```

```
gcc -o dzasumEX dzasumEX.f -lgfortran dzasum.o dcabs1.o && ./dzasumEX
```

```
before:
```

```
a(1)=C( 1.000  2.000)
```

```
a(2)=C(-3.000  4.000)
```

```
a(3)=C( 5.000 -6.000)
```

```
should be 21:
```

```
b=21.000
```

```
should be 14:
```

```
b=14.000
```

```
should be 0:
```

```
b= 0.000
```

Spad represents complex numbers as a pair where the car is the real part and the cons is the imaginary part. BLAS wants a complex number in fortran format. So we have a design choice to make. Either we could write all of the BLAS code using Spad internal representation or we could follow the BLAS code standard. I've decided to follow the BLAS code standard so we need to create thunks to do the data translation.

— BLAS 1 dzasum —

```
(defun dzasumSpad (n zx incx)
; Tim Daly May 7, 2012
  (let (result vec)
    (dotimes (i (length zx))
      (push (complex (car (svref zx i)) (cdr (svref zx i))) result))
    (setq vec (make-array (length result) :initial-contents (nreverse result)))
    (dzasum n vec incx)))

(defun dzasum (n zx incx)
; Tim Daly May 7, 2012
  (declare (type (simple-array (complex double-float) (*)) zx)
    (type fixnum incx n))
  (let ((ix 0) (stemp 0.0d0) (dzasum 0.0d0) (limit (length zx)))
    (declare (type (double-float) dzasum stemp)
      (type fixnum ix limit))
    (when (and (> n 0) (> incx 0))
      (do ((i 0 (1+ i)))
        ((or (>= i n) (>= ix limit)) nil)
        (setf dzasum (+ dzasum (the double-float (dcabs1 (svref zx ix)))))
        (setf ix (+ ix incx))))
    dzasum))
```

```
--E 3
```

```
--S 4 of 8
```

```
dznrm2(3,a,1) -- should be 11.269
```

```
--R
```

```
--R (4) 11.269427669584644
```

```
--R
```

Type: DoubleFloat

```
--E 4
```

```
--S 5 of 8
```

```
dznrm2(3,a,-1) -- should be 0.0
```

```
--R
```

```
--R (5) 0.
```

```
--R
```

Type: DoubleFloat

```
--E 5
```

```
--S 6 of 8
```

```
dznrm2(-3,a,-1) -- should be 0.0
```

```
--R
```

```
--R (6) 0.
```

```
--R
```

Type: DoubleFloat

```
--E 6
```

```
--S 7 of 8
```

```
dznrm2(1,a,1) -- should be 5.0
```

```
--R
```

```
--R (7) 5.
```

```
--R
```

Type: DoubleFloat

```
--E 7
```

```
--S 8 of 8
```

```
dznrm2(1,a,2) -- should be 5.0
```

```
--R
```

```
--R (8) 5.
```

```
--R
```

Type: DoubleFloat

```
--E 8
```

```
)spool
```

```
)lisp (bye)
```

— dznrm2.help —

```
=====
```

```
dznrm2 examples
```

```
=====
```

dznrm2 computes the complex norm of the input vector.

For each element it computes the sum
 $a.i \cdot \text{conjugate}(a.i)$
 for all i . Then it returns the square root of the final sum.
 If the count or increment are negative, the result is 0.0

```
a:PRIMARR(COMPLEX(DFLOAT)):=_
  [[3.+4.*%i, -4.+5.*%i, 5.+6.*%i, 7.-8.*%i, -9.-2.*%i]]

  [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]

dznrm2(5,a,1)

18.027756377319946

dznrm2(3,a,2)

13.076696830622021

dznrm2(3,a,1)

11.269427669584644

dznrm2(3,a,-1)

0.

dznrm2(-3,a,-1)

0.

dznrm2(1,a,1)

5.

dznrm2(1,a,2)

5.
```

```
=====
Man Page Details
=====
```

NAME

DZNRM2 - BLAS level one, computes the Euclidean norm of a vector

SYNOPSIS

DOUBLE PRECISION FUNCTION DZNRM2 (n, x, incx)

INTEGER

n, incx

DOUBLE COMPLEX x

DESCRIPTION

DZNRM2 computes the Euclidean (L2) norm of a double precision complex vector:

$$DZNRM2 \leftarrow \sqrt{|x|^2}$$

where x is a double precision complex vector.

ARGUMENTS

n INTEGER (input)
 Number of elements in the operand vector.

x DOUBLE COMPLEX (input)
 Array of dimension (n-1) * |incx| + 1.
 Array x contains the operand vector.

incx INTEGER (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

RETURN VALUES

DZNRM2 DOUBLE PRECISION Result (Euclidean norm). (output)
 If n <= 0, DZNRM2 is set to 0d0.

NOTES

When working backward (incx < 0), DZNRM2 starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

— dznrm2.f —

```
DOUBLE PRECISION FUNCTION DZNRM2( N, X, INCX )
*   .. Scalar Arguments ..
      INTEGER                INCX, N
*   .. Array Arguments ..
      COMPLEX*16             X( * )
*
*   ..
*
* DZNRM2 returns the euclidean norm of a vector via the function
```

```

*   name, so that
*
*   DZNRM2 := sqrt( conjg( x' ) * x )
*
*
*
*   -- This version written on 25-October-1982.
*   Modified on 14-October-1993 to inline the call to ZLASSQ.
*   Sven Hammarling, Nag Ltd.
*
*
*   .. Parameters ..
*   DOUBLE PRECISION      ONE          , ZERO
*   PARAMETER              ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*   .. Local Scalars ..
*   INTEGER                IX
*   DOUBLE PRECISION       NORM, SCALE, SSQ, TEMP
*   .. Intrinsic Functions ..
*   INTRINSIC              ABS, DIMAG, DBLE, SQRT
*
*   .. Executable Statements ..
*   IF( N.LT.1 .OR. INCX.LT.1 )THEN
*       NORM = ZERO
*   ELSE
*       SCALE = ZERO
*       SSQ   = ONE
*
*       The following loop is equivalent to this call to the LAPACK
*       auxiliary routine:
*       CALL ZLASSQ( N, X, INCX, SCALE, SSQ )
*
*       DO 10, IX = 1, 1 + ( N - 1 ) * INCX, INCX
*           IF( DBLE( X( IX ) ).NE.ZERO )THEN
*               TEMP = ABS( DBLE( X( IX ) ) )
*               IF( SCALE.LT.TEMP )THEN
*                   SSQ   = ONE  + SSQ * ( SCALE / TEMP ) ** 2
*                   SCALE = TEMP
*               ELSE
*                   SSQ   = SSQ   + ( TEMP / SCALE ) ** 2
*               END IF
*           END IF
*           IF( DIMAG( X( IX ) ).NE.ZERO )THEN
*               TEMP = ABS( DIMAG( X( IX ) ) )
*               IF( SCALE.LT.TEMP )THEN
*                   SSQ   = ONE  + SSQ * ( SCALE / TEMP ) ** 2
*                   SCALE = TEMP
*               ELSE
*                   SSQ   = SSQ   + ( TEMP / SCALE ) ** 2
*               END IF
*           END IF
*       10 CONTINUE

```

```

        NORM = SCALE * SQRT( SSQ )
    END IF
*
    DZNRM2 = NORM
    RETURN
*
*   End of DZNRM2.
*
END

```

— dznrm2 example —

```

program dznrm2EX
*   Tim Daly May 8, 2012
*   unit tests for BLAS dznrm2
    complex*16 a(5)
    double precision c
    a(1) = ( 3.0D0, 4.0D0)
    a(2) = (-4.0D0, 5.0D0)
    a(3) = ( 5.0D0, 6.0D0)
    a(4) = ( 7.0D0,-8.0D0)
    a(5) = (-9.0D0,-2.0D0)
    write(6,100)a(1),a(2),a(3),a(4),a(5)
100  format("a=(/  (",f6.3,f6.3,")",("f6.3,f6.3"),("f6.3,f6.3"),"/,
C      "      (",f6.3,f6.3,")",("f6.3,f6.3") /)")
    c=dznrm2(5,a,1)
    write(6,200)c
200  format("should be 18.028",/,"c=",f6.3)
    c=dznrm2(3,a,2)
    write(6,201)c
201  format("should be 13.077",/,"c=",f6.3)
    c=dznrm2(3,a,1)
    write(6,202)c
202  format("should be 11.269",/,"c=",f6.3)
    c=dznrm2(3,a,-1)
    write(6,203)c
203  format("should be 0.000",/,"c=",f6.3)
    c=dznrm2(-3,a,-1)
    write(6,204)c
204  format("should be 0.000",/,"c=",f6.3)
    c=dznrm2(1,a,1)
    write(6,205)c
205  format("should be 5.000",/,"c=",f6.3)
    c=dznrm2(1,a,2)
    write(6,206)c
206  format("should be 5.000",/,"c=",f6.3)

```

```

stop
end

```

```

gcc -o dznrm2EX dznrm2EX.f -lgfortran dznrm2.o && ./dznrm2EX
a=(/ ( 3.000 4.000),(-4.000 5.000),( 5.000 6.000),
      ( 7.000-8.000),(-9.000-2.000) /)
should be 18.028
c=18.028
should be 13.077
c=13.077
should be 11.269
c=11.269
should be 0.000
c= 0.000
should be 0.000
c= 0.000
should be 5.000
c= 5.000
should be 5.000
c= 5.000

```

Spad represents complex numbers as a pair where the car is the real part and the cons is the imaginary part. BLAS wants a complex number in fortran format. So we have a design choice to make. Either we could write all of the BLAS code using Spad internal representation or we could follow the BLAS code standard. I've decided to follow the BLAS code standard so we need to create thunks to do the data translation.

NOTE: The Axiom internal representation of PRIMARR(COMPLEX(FLOAT)) is an array where each array element AE is

```
AE = (cons (cons (therealpart 0)) (cons (theimagpart 0)))
```

so we need (caar AE) and (cadr AE) to get the real and imag parts.

— BLAS 1 dznrm2 —

```

(defun dznrm2Spad (n zx incx)
; Tim Daly May 8, 2012
  (let (result vec)
    (dotimes (i (length zx))
      (push (complex (car (svref zx i)) (cdr (svref zx i))) result))
    (setq vec (make-array (length result) :initial-contents (nreverse result)))
    (dznrm2 n vec incx)))

(defun dznrm2 (n x incx)
; Tim Daly May 8, 2012

```

```

(declare (type (simple-array (complex double-float) (*)) x)
  (type fixnum incx n))
(let ((scale 0.0d0) (ssq 0.0d0) (temp 0.0d0)
      (dznrm2 0.0d0) (limitx (min (* n incx) (length x))))
  (declare (type fixnum limitx) (type (double-float) scale ssq temp dznrm2))
  (when (and (>= n 1) (>= incx 1))
    (do ((ix 0 (the fixnum (+ ix incx))))
        ((>= ix limitx))
      (when (/= (the double-float (realpart
                                   (the (complex double-float) (svref x ix))))
                0.0d0)
        (setf temp
              (the double-float (abs
                                   (the double-float (realpart
                                                         (the (complex double-float) (svref x ix)))))))
        (cond
          ((< scale temp)
           (setf ssq (+ 1.0d0
                       (the double-float (*
                                           ssq
                                           (the double-float (*
                                                                 (the double-float (/ scale temp))
                                                                 (the double-float (/ scale temp))))))))
           (setf scale temp))
          (t
           (setf ssq (+ ssq (*
                           (the double-float (/ temp scale))
                           (the double-float (/ temp scale)))))))
        (when (/= (the double-float (imagpart
                                      (the (complex double-float) (svref x ix)))) 0.0d0)
          (setf temp
                (the double-float (abs
                                      (the double-float (imagpart
                                                            (the (complex double-float) (svref x ix)))))))
          (cond
            ((< scale temp)
             (setf ssq (+ 1.0d0 (* ssq (*
                                       (the double-float (/ scale temp))
                                       (the double-float (/ scale temp))))))
             (setf scale temp))
            (t
             (setf ssq (+ ssq (*
                             (the double-float (/ temp scale))
                             (the double-float (/ temp scale)))))))
          (setf dznrm2 (the double-float (* scale (the double-float (sqrt ssq)))))
          (setf dznrm2))

```

— BLAS 1 dznrm2 lisp test —

```

(load "dznrm2.lisp")
(setq a (vector #C(3.0d0 4.0d0) #C(-4.0d0 5.0d0) #C(5.0d0 6.0d0)
               #C(7.0d0 -8.0d0) #C(-9.0d0 -2.0d0)))
; #(#C(3.0 4.0) #C(-4.0 5.0) #C(5.0 6.0) #C(7.0 -8.0) #C(-9.0 -2.0))
(dznrm2 5 a 1)
; 18.027756377319946
(dznrm2 3 a 2)
; 13.076696830622023
(dznrm2 3 a 1)
; 11.269427669584648
(dznrm2 3 a -1)
; 0.0
(dznrm2 -3 a -1)
; 0.0
(dznrm2 1 a 1)
; 5.0
(dznrm2 1 a 2)
; 5.0

```

icamax BLAS

— icamax.input —

```

)set break resume
)sys rm -f icamax.output
)spool icamax.output
)set message test on
)set message auto off
)clear all

--S 1 of 7
a:PRIMARR(COMPLEX(FLOAT))
--R
--E 1
Type: Void

--S 2 of 7
a:=[[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]
--R
--R (9) [3.0 + 4.0 %i,- 4.0 + 5.0 %i,5.0 + 6.0 %i,7.0 - 8.0 %i,- 9.0 - 2.0 %i]
--R
--R Type: PrimitivesArray(Complex(Float))
--E 2

```

```

--S 3 of 7
icamax(5,a,1) -- should be 3
--R
--R (10) 3
--R
--R                                         Type: PositiveInteger
--E 3

--S 4 of 7
icamax(0,a,1) -- should be -1
--R
--R (11) - 1
--R
--R                                         Type: Integer
--E 4

--S 5 of 7
icamax(5,a,-1) -- should be -1
--R
--R (12) - 1
--R
--R                                         Type: Integer
--E 5

--S 6 of 7
icamax(3,a,1) -- should be 2
--R
--R (13) 2
--R
--R                                         Type: PositiveInteger
--E 6

--S 7 of 7
icamax(3,a,2) -- should be 1
--R
--R (14) 1
--R
--R                                         Type: PositiveInteger
--E 7

)spool
)lisp (bye)

```

— icamax.help —

```

=====
icamax examples
=====

```

```
a:PRIMARR(COMPLEX(FLOAT))
```

```
a:=[[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]
```

```
[3.0 + 4.0 %i,- 4.0 + 5.0 %i,5.0 + 6.0 %i,7.0 - 8.0 %i,- 9.0 - 2.0 %i]
```

Note that Axiom arrays are 0-based

```
icamax(5,a,1) -- should be 3
```

```
3
```

If the arguments are not valid, return an invalid array index

The count must be greater than 0.

```
icamax(0,a,1) -- should be -1
```

```
- 1
```

The increment must be positive

```
icamax(5,a,-1) -- should be -1
```

```
- 1
```

```
icamax(3,a,1) -- should be 2
```

```
2
```

```
icamax(3,a,2) -- should be 1
```

```
1
```

```
=====
Man Page Details
=====
```

NAME

ICAMAX - BLAS level one, maximum index function

SYNOPSIS

```
INTEGER FUNCTION ICAMAX ( n, x, incx )
```

```
INTEGER                n, incx
```

```
COMPLEX                x
```

DESCRIPTION

ICAMAX searches a complex vector for the first occurrence of the maximum absolute value.

ICAMAX determines the first index i such that

$$|\text{Real}(x_i)| + |\text{Imag}(x_i)| = \text{MAX}(|\text{Real}(x_j)| + |\text{Imag}(x_j)|): j = 1, \dots, n$$

where x_j is an element of a complex vector.

ARGUMENTS

n INTEGER. (input)
Number of elements to process in the vector to be searched. If $n \leq 0$, these routines return 0.

x COMPLEX. (input)
Array of dimension $(n-1) * |\text{incx}| + 1$.
Array x contains the vector to be searched.

incx INTEGER. (input)
Increment between elements of x .

RETURN VALUES

ICAMAX INTEGER. (output)
Return the first index of the maximum absolute value of vector x . The vector x has length n and increment incx .

NOTES

When working backward ($\text{incx} < 0$), each routine starts at the end of the vector and moves backward, as follows:

$$x(1-\text{incx} * (n-1)), x(1-\text{incx} * (n-2)), \dots, x(1)$$

The largest absolute value is:

$$\text{ABS}(x(1+(\text{index}-1) * \text{incx})) \text{ when } \text{incx} > 0$$

$$\text{ABS}(x(1+(n-\text{index}) * |\text{incx}|)) \text{ when } \text{incx} < 0$$

— icamax.f —

integer function icamax(n,cx,incx)

c
c finds the index of element having max. absolute value.
c jack dongarra, linpack, 3/11/78.
c modified 3/93 to return if incx .le. 0.
c modified 12/3/93, array(1) declarations changed to array(*)
c
c complex cx(*)

```

      real smax
      integer i,incx,ix,n
      complex zdum
      real cabs1
      cabs1(zdum) = abs(real(zdum)) + abs(aimag(zdum))
c
      icamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      icamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
      ix = 1
      smax = cabs1(cx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(cabs1(cx(ix)).le.smax) go to 5
         icamax = i
         smax = cabs1(cx(ix))
      5   ix = ix + incx
      10 continue
      return
c
c      code for increment equal to 1
c
      20 smax = cabs1(cx(1))
      do 30 i = 2,n
         if(cabs1(cx(i)).le.smax) go to 30
         icamax = i
         smax = cabs1(cx(i))
      30 continue
      return
      end

```

— icamax example —

```

program icamaxEX
*   Tim Daly May 17, 2012
*   unit tests for BLAS icamax
      complex a(5)
      integer n
      a(1) = ( 1.0, 2.0)
      a(2) = (-2.0, 2.0)
      a(3) = ( 2.0,-1.0)

```

```

      a(4) = ( 2.0,-3.0)
      a(5) = (-1.0,-0.0)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
100  format("a=(/ (",f6.3,f6.3,")",(",f6.3,f6.3")",(",f6.3,f6.3")",",/,
C      "      (",f6.3,f6.3,")",(",f6.3,f6.3") /)")
      n=icamax(5,a,1)
      write(6,200)n
200  format("should be 4",/,"n=",i3)
      n=icamax(3,a,1)
      write(6,201)n
201  format("should be 2",/,"n=",i3)
      n=icamax(0,a,1)
      write(6,202)n
202  format("should be 0",/,"n=",i3)
      n=icamax(-5,a,1)
      write(6,203)n
203  format("should be 0",/,"n=",i3)
      n=icamax(5,a,-1)
      write(6,204)n
204  format("should be 0",/,"n=",i3)
      n=icamax(5,a,2)
      write(6,205)n
205  format("should be 1",/,"n=",i3)
      n=icamax(1,a,0)
      write(6,206)n
206  format("should be 0",/,"n=",i3)
      n=icamax(1,a,-1)
      write(6,207)n
207  format("should be 0",/,"n=",i3)
      n=icamax(1,a,5)
      write(6,208)n
208  format("should be 1",/,"n=",i3)
      a(1) = ( 1.0, 2.0)
      a(2) = (-2.0, 2.0)
      a(3) = ( 2.0,-1.0)
      a(4) = (-2.0,-3.0)
      a(5) = (-1.0,-0.0)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
      n=icamax(5,a,1)
      write(6,209)n
209  format("should be 4",/,"n=",i3)
      stop
      end

```

```

gcc -o icamaxEX icamaxEX.f -lgfortran icamax.o && ./icamaxEX
a=(/ ( 1.000 2.000),(-2.000 2.000),( 2.000-1.000),
      ( 2.000-3.000),(-1.000-0.000) /)
should be 4

```

```

n= 4
should be 2
n= 2
should be 0
n= 0
should be 0
n= 0
should be 0
n= 0
should be 1
n= 1
should be 0
n= 0
should be 0
n= 0
should be 1
n= 1
a=(/ ( 1.000 2.000),(-2.000 2.000),( 2.000-1.000),
      (-2.000-3.000),(-1.000-0.000) /)
should be 4
n= 4

```

Spad represents complex numbers as a pair where the car is the real part and the cons is the imaginary part. BLAS wants a complex number in fortran format. So we have a design choice to make. Either we could write all of the BLAS code using Spad internal representation or we could follow the BLAS code standard. I've decided to follow the BLAS code standard so we need to create thunks to do the data translation.

NOTE: The Axiom internal representation of PRIMARR(COMPLEX(FLOAT)) is an array where each array element AE is

```
AE = (cons (cons (therealpart 0)) (cons (theimagpart 0)))
```

so we need (caar AE) and (cadr AE) to get the real and imag parts.

NOTE: Array indexing in Axiom is 0-based but Fortran is 1-based so the results are displaced by 1.

— BLAS 1 icamax —

```

(defun icamaxSpad (n zx incx)
; Tim Daly May 13, 2012
(let (result vec)
  (dotimes (i (length zx))
    (push (complex
      (coerce (caar (svref zx i)) 'single-float)
      (coerce (cadr (svref zx i)) 'single-float)) result))
  (setq vec (make-array (length result) :initial-contents (nreverse result)
    :element-type '(complex single-float)))

```

```

(icamax n vec incx))

(defun icamax (n cx incx)
  (declare (type (simple-array (complex single-float) (*)) cx)
           (type fixnum incx n))
  (labels (
    (cabs1 (zdum)
      (the single-float
        (+ (the single-float (abs (the single-float (realpart zdum))))
           (the single-float (abs (the single-float (imagpart zdum)))))))
    (declare (ftype (function (complex single-float) single-float) cabs1))
    (let ((ix 0) (smax 0.0f0) (icamax -1) (limit (length cx)))
      (declare (type (single-float) smax) (type fixnum icamax ix limit))
      (when (and (>= n 1) (> incx 0))
        (setq icamax 0)
        (setf smax (the single-float (cabs1 (svref cx 0))))
        (setf ix (the fixnum (+ ix incx)))
        (do ((i 1 (+ i 1)))
          ((or (>= ix limit) (> i n)))
            (when (> (cabs1 (the (complex single-float) (svref cx ix)))
                    smax)
              (setf icamax i)
              (setf smax (cabs1 (svref cx ix))))
            (setf ix (the fixnum (+ ix incx))))))
        icamax)))

```

— BLAS 1 icamax lisp test —

```

(load "icamax.lisp")
(setq a
  (vector #C(1.0 2.0) #C(-2.0 2.0) #C(2.0 -1.0) #C(2.0 -3.0) #C(-1.0 -0.0)))
; #C(1.0 2.0) #C(-2.0 2.0) #C(2.0 -1.0) #C(2.0 -3.0) #C(-1.0 0.0)
(icamax 5 a 1)
; 3
(icamax 3 a 1)
; 1
(icamax 0 a 1)
; -1
(icamax -5 a 1)
; -1
(icamax 5 a -1)
; -1
(icamax 5 a 2)
; 0
(icamax 1 a 0)
; -1

```

```

(icamax 1 a -1)
; -1
(icamax 1 a 5)
; 0
(setq a
  (vector #C(1.0 2.0) #C(-2.0 2.0) #C(2.0 -1.0) #C(-2.0 -3.0) #C(-1.0 -0.0)))
; #(#C(1.0 2.0) #C(-2.0 2.0) #C(2.0 -1.0) #C(-2.0 -3.0) #C(-1.0 0.0))
(icamax 5 a 1)
; 3

```

idamax BLAS

— idamax.input —

```

)set break resume
)sys rm -f idamax.output
)spool idamax.output
)set message test on
)set message auto off
)clear all

--S 1 of 11
a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
--R
--R (1) [3.,4.,- 3.,5.,- 1.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 11
idamax(5,a,1) -- should be 3
--R
--R (2) 3
--R
--R                                          Type: PositiveInteger
--E 2

--S 3 of 11
idamax(3,a,1) -- should be 1
--R
--R (3) 1
--R
--R                                          Type: PositiveInteger
--E 3

--S 4 of 11
idamax(0,a,1) -- should be -1

```

```

--R
--R (4) - 1
--R
--R                                         Type: Integer
--E 4

--S 5 of 11
idamax(-5,a,1) -- should be -1
--R
--R (5) - 1
--R
--R                                         Type: Integer
--E 5

--S 6 of 11
idamax(5,a,-1) -- should be -1
--R
--R (6) - 1
--R
--R                                         Type: Integer
--E 6

--S 7 of 11
idamax(5,a,2) -- should be 0
--R
--R (7) 0
--R
--R                                         Type: NonNegativeInteger
--E 7

--S 8 of 11
idamax(1,a,0) -- should be -1
--R
--R (8) - 1
--R
--R                                         Type: Integer
--E 8

--S 9 of 11
idamax(1,a,-1) -- should be -1
--R
--R (9) - 1
--R
--R                                         Type: Integer
--E 9

--S 10 of 11
a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
--R
--R (10) [3.,4.,- 3.,- 5.,- 1.]
--R
--R                                         Type: PrimitiveArray(DoubleFloat)
--E 10

--S 11 of 11
idamax(5,a,1) -- should be 3
--R

```

```
--R (11) 3
--R
--E 11
```

Type: PositiveInteger

```
)spool
)lisp (bye)
```

— idamax.help —

```
=====
idamax examples
=====
```

```
a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
```

```
[3.,4.,- 3.,5.,- 1.]
```

Note that Axiom arrays are 0-based.

```
idamax(5,a,1) -- should be 3
```

```
3
```

```
idamax(3,a,1) -- should be 1
```

```
1
```

The count and increment must both be positive numbers

```
idamax(0,a,1) -- should be -1
```

```
- 1
```

```
idamax(-5,a,1) -- should be -1
```

```
- 1
```

```
idamax(5,a,-1) -- should be -1
```

```
- 1
```

```
idamax(5,a,2) -- should be 0
```

```
0
```

```
idamax(1,a,0) -- should be -1
```


- 1

idamax(1,a,-1) -- should be -1

- 1

The comparison for maximum uses the absolute value

a:PRIMARR(DFLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]

[3.,4.,- 3.,- 5.,- 1.]

idamax(5,a,1) -- should be 3

3

Man Page Details

NAME

IDAMAX - BLAS level one, maximum index function

SYNOPSIS

INTEGER FUNCTION IDAMAX (n, x, incx)

INTEGER n, incx

DOUBLE PRECISION x

DESCRIPTION

IDAMAX searches a double precision vector for the first occurrence of the the maximum absolute value. The vector x has length n and increment incx.

ARGUMENTS

n INTEGER. (input)
Number of elements to process in the vector to be searched. If n <= 0, these routines return 0.

x DOUBLE PRECISION. (input)
Array of dimension (n-1) * |incx| + 1.
Array x contains the vector to be searched.

incx INTEGER. (input)
Increment between elements of x.

RETURN VALUES

IDAMAX INTEGER. (output)

Return the first index of the maximum absolute value of vector x . The vector x has length n and increment $incx$.

NOTES

When working backward ($incx < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

The largest absolute value is:

$ABS(x(1+(index-1) * incx))$ when $incx > 0$

$ABS(x(1+(n-index) * |incx|))$ when $incx < 0$

—————

— idamax.f —

```
integer function idamax(n,dx,incx)
c
c finds the index of element having max. absolute value.
c jack dongarra, linpack, 3/11/78.
c modified 3/93 to return if incx .le. 0.
c modified 12/3/93, array(1) declarations changed to array(*)
c
double precision dx(*),dmax
integer i,incx,ix,n
c
idamax = 0
if( n.lt.1 .or. incx.le.0 ) return
idamax = 1
if(n.eq.1)return
if(incx.eq.1)go to 20
c
c code for increment not equal to 1
c
ix = 1
dmax = dabs(dx(1))
ix = ix + incx
do 10 i = 2,n
    if(dabs(dx(ix)).le.dmax) go to 5
    idamax = i
    dmax = dabs(dx(ix))
5 ix = ix + incx
10 continue
return
c
```

```

c      code for increment equal to 1
c
20  dmax = dabs(dx(1))
    do 30 i = 2,n
        if(dabs(dx(i)).le.dmax) go to 30
        idamax = i
        dmax = dabs(dx(i))
30  continue
    return
end

```

— idamaxEX example —

```

program idamaxEX
*      Tim Daly May 16, 2012
*      unit tests for BLAS idamax
double precision a(5)
integer n
a = (/ 3.0, 4.0, -3.0, 5.0, -1.0 /)
write(6,100)a(1),a(2),a(3),a(4),a(5)
100  format("a=(/ (/ ",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3," /)")
n=idamax(5,a,1)
write(6,200)n
200  format("should be 4",/,"n=",i3)
n=idamax(3,a,1)
write(6,201)n
201  format("should be 2",/,"n=",i3)
n=idamax(0,a,1)
write(6,202)n
202  format("should be 0",/,"n=",i3)
n=idamax(-5,a,1)
write(6,203)n
203  format("should be 0",/,"n=",i3)
n=idamax(5,a,-1)
write(6,204)n
204  format("should be 0",/,"n=",i3)
n=idamax(5,a,2)
write(6,205)n
205  format("should be 1",/,"n=",i3)
n=idamax(1,a,0)
write(6,206)n
206  format("should be 0",/,"n=",i3)
n=idamax(1,a,-1)
write(6,207)n
207  format("should be 0",/,"n=",i3)
n=idamax(1,a,5)

```

```

write(6,208)n
208 format("should be 1",/,"n=",i3)
a = (/ 3.0, 4.0, -3.0, -5.0, -1.0 /)
write(6,100)a(1),a(2),a(3),a(4),a(5)
n=idamax(5,a,1)
write(6,209)n
209 format("should be 4",/,"n=",i3)
stop
end

```

—————→

```

gcc -o idamaxEX idamaxEX.f -lgfortran idamax.o && ./idamaxEX
a=(/ (/ 3.000  4.000 -3.000  5.000 -1.000 /)
should be 4
n=  4
should be 2
n=  2
should be 0
n=  0
should be 0
n=  0
should be 0
n=  0
should be 1
n=  1
should be 0
n=  0
should be 0
n=  0
should be 1
n=  1
a=(/ (/ 3.000  4.000 -3.000 -5.000 -1.000 /)
should be 4
n=  4

```

— BLAS 1 idamax —

```

(defun idamax (n dx incx)
  (declare (type (simple-array double-float (*)) dx)
           (type fixnum incx n))
  (let ((ix 0) (dmax 0.0) (idamax -1) (limit (length dx)))
    (declare (type (double-float) dmax) (type fixnum idamax ix limit))
    (when (and (>= n 1) (> incx 0))
      (setq idamax 0)
      (setf dmax (the double-float (abs (the double-float (svref dx 0)))))
      (setf ix (the fixnum (+ ix incx)))
      (do ((i 1 (+ i 1)))

```

```

      ((or (>= ix limit) (>= i n)))
      (when (> (the double-float (abs (the double-float (svref dx ix)))) dxmax)
        (setf idamax i)
        (setf dxmax (the double-float (abs (the double-float (svref dx ix)))))
        (setf ix (the fixnum (+ ix incx)))))
      idamax))

```

— BLAS 1 idamax lisp test —

```

(load "idamax.lisp")
(setq a (vector 3.0d0 4.0d0 -3.0d0 5.0d0 -1.0d0))
; #(3.0 4.0 -3.0 5.0 -1.0)
(idamax 5 a 1)
; 3
(idamax 3 a 1)
; 1
(idamax 0 a 1)
; -1
(idamax -5 a 1)
; -1
(idamax 5 a -1)
; -1
(idamax 5 a 2)
; 0
(idamax 1 a 0)
; -1
(idamax 1 a -1)
; -1
(idamax 1 a 5)
; 0
(setq a (vector 3.0d0 4.0d0 -3.0d0 -5.0d0 -1.0d0))
; #(3.0 4.0 -3.0 -5.0 -1.0)
(idamax 5 a 1)
; 3

```

isamax BLAS

— isamax.input —

```

)set break resume
)sys rm -f isamax.output

```

```

)spool isamax.output
)set message test on
)set message auto off
)clear all

--S 1 of 11
a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]
--R
--R (1) [3.,4.,- 3.,5.,- 1.]
--R
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 1

--S 2 of 11
isamax(5,a,1) -- should be 3
--R
--R (2) 3
--R
--R                                          Type: PositiveInteger
--E 2

--S 3 of 11
isamax(3,a,1) -- should be 1
--R
--R (3) 1
--R
--R                                          Type: PositiveInteger
--E 3

--S 4 of 11
isamax(0,a,1) -- should be -1
--R
--R (4) - 1
--R
--R                                          Type: Integer
--E 4

--S 5 of 11
isamax(-5,a,1) -- should be -1
--R
--R (5) - 1
--R
--R                                          Type: Integer
--E 5

--S 6 of 11
isamax(5,a,-1) -- should be -1
--R
--R (6) - 1
--R
--R                                          Type: Integer
--E 6

--S 7 of 11
isamax(5,a,2) -- should be 0
--R

```

```

--R (7) 0
--R                                          Type: NonNegativeInteger
--E 7

--S 8 of 11
isamax(1,a,0) -- should be -1
--R
--R (8) - 1
--R                                          Type: Integer
--E 8

--S 9 of 11
isamax(1,a,-1) -- should be -1
--R
--R (9) - 1
--R                                          Type: Integer
--E 9

--S 10 of 11
a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
--R
--R (10) [3.,4.,- 3.,- 5.,- 1.]
--R                                          Type: PrimitiveArray(DoubleFloat)
--E 10

--S 11 of 11
isamax(5,a,1) -- should be 3
--R
--R (11) 3
--R                                          Type: PositiveInteger
--E 11

)spool
)lisp (bye)

```

— isamax.help —

```

=====
isamax examples
=====

```

```

a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, 5.0, -1.0]]

```

```

[3.,4.,- 3.,5.,- 1.]

```

Note that Axiom arrays are 0-based.

```
isamax(5,a,1)  -- should be 3
```

```
3
```

```
isamax(3,a,1)  -- should be 1
```

```
1
```

The count and increment must both be positive numbers

```
isamax(0,a,1)  -- should be -1
```

```
- 1
```

```
isamax(-5,a,1) -- should be -1
```

```
- 1
```

```
isamax(5,a,-1) -- should be -1
```

```
- 1
```

```
isamax(5,a,2)  -- should be 0
```

```
0
```

```
isamax(1,a,0)  -- should be -1
```

```
- 1
```

```
isamax(1,a,-1) -- should be -1
```

```
- 1
```

The comparison for maximum uses the absolute value

```
a:PRIMARR(FLOAT):=[[3.0, 4.0, -3.0, -5.0, -1.0]]
```

```
[3.,4.,- 3.,- 5.,- 1.]
```

```
isamax(5,a,1)  -- should be 3
```

```
3
```

```
=====
Man Page Details
=====
```

NAME

ISAMAX - BLAS level one, maximum index function

SYNOPSIS

```

      INTEGER FUNCTION ISAMAX ( n, x, incx )

      INTEGER          n, incx

      REAL             x

```

DESCRIPTION

ISAMAX searches a real vector for the first occurrence of the the maximum absolute value. The vector x has length n and increment incx.

ISAMAX returns the first index i such that

$$|x_i| = \max_j |x_j| : j = 1, \dots, n$$

where x_j is an element of a real vector.

ARGUMENTS

n INTEGER. (input)
 Number of elements to process in the vector to be searched. If
 n <= 0, these routines return 0.

x REAL. (input)
 Array of dimension (n-1) * |incx| + 1.
 Array x contains the vector to be searched.

incx INTEGER. (input)
 Increment between elements of x.

RETURN VALUES

ISAMAX INTEGER. (output)
 Return the first index of the maximum absolute value of vector
 x. The vector x has length n and increment incx.

NOTES

When working backward (incx < 0), each routine starts at the end of the vector and moves backward, as follows:

$$x(1-\text{incx} * (n-1)), x(1-\text{incx} * (n-2)), \dots, x(1)$$

The largest absolute value is:

$$\text{ABS} (x(1+(\text{index}-1) * \text{incx})) \text{ when } \text{incx} > 0$$

$$\text{ABS} (x(1+(n-\text{index}) * |\text{incx}|)) \text{ when } \text{incx} < 0$$

— isamax.f —

```

integer function isamax(n,sx,incx)
c
c  finds the index of element having max. absolute value.
c  jack dongarra, linpack, 3/11/78.
c  modified 3/93 to return if incx .le. 0.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
  real sx(*),smax
  integer i,incx,ix,n
c
  isamax = 0
  if( n.lt.1 .or. incx.le.0 ) return
  isamax = 1
  if(n.eq.1)return
  if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
  ix = 1
  smax = abs(sx(1))
  ix = ix + incx
  do 10 i = 2,n
    if(abs(sx(ix)).le.smax) go to 5
    isamax = i
    smax = abs(sx(ix))
  5   ix = ix + incx
 10 continue
  return
c
c      code for increment equal to 1
c
 20 smax = abs(sx(1))
  do 30 i = 2,n
    if(abs(sx(i)).le.smax) go to 30
    isamax = i
    smax = abs(sx(i))
 30 continue
  return
end

```

— isamaxEX example —

```

program isamaxEX

```

```

*      Tim Daly May 19, 2012
*      unit tests for BLAS isamax
      real a(5)
      integer n
      a = (/ 3.0, 4.0, -3.0, 5.0, -1.0 /)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
100    format("a=(/ (/ ",f6.3," ",f6.3," ",f6.3," ",f6.3," ",f6.3," /)")
      n=isamax(5,a,1)
      write(6,200)n
200    format("should be 4",/,"n=",i3)
      n=isamax(3,a,1)
      write(6,201)n
201    format("should be 2",/,"n=",i3)
      n=isamax(0,a,1)
      write(6,202)n
202    format("should be 0",/,"n=",i3)
      n=isamax(-5,a,1)
      write(6,203)n
203    format("should be 0",/,"n=",i3)
      n=isamax(5,a,-1)
      write(6,204)n
204    format("should be 0",/,"n=",i3)
      n=isamax(5,a,2)
      write(6,205)n
205    format("should be 1",/,"n=",i3)
      n=isamax(1,a,0)
      write(6,206)n
206    format("should be 0",/,"n=",i3)
      n=isamax(1,a,-1)
      write(6,207)n
207    format("should be 0",/,"n=",i3)
      n=isamax(1,a,5)
      write(6,208)n
208    format("should be 1",/,"n=",i3)
      a = (/ 3.0, 4.0, -3.0, -5.0, -1.0 /)
      write(6,100)a(1),a(2),a(3),a(4),a(5)
      n=isamax(5,a,1)
      write(6,209)n
209    format("should be 4",/,"n=",i3)
      stop
      end

```

```

gcc -o isamaxEX isamaxEX.f -lgfortran isamax.o && ./isamaxEX
a=(/ (/ 3.000 4.000 -3.000 5.000 -1.000 /)
should be 4
n= 4
should be 2
n= 2

```

```

should be 0
n= 0
should be 0
n= 0
should be 0
n= 0
should be 1
n= 1
should be 0
n= 0
should be 0
n= 0
should be 1
n= 1
a=(/ (/ 3.000 4.000 -3.000 -5.000 -1.000 /)
should be 4
n= 4

```

Spad represents complex numbers as a pair where the car is the real part and the cons is the imaginary part. BLAS wants a complex number in fortran format. So we have a design choice to make. Either we could write all of the BLAS code using Spad internal representation or we could follow the BLAS code standard. I've decided to follow the BLAS code standard so we need to create thunks to do the data translation.

— BLAS 1 isamax —

```

(defun isamaxSpad (n zx incx)
; Tim Daly May 19, 2012
(let (result vec)
  (dotimes (i (length zx)) (push (float (car (svref zx i))) result))
  (setq vec (make-array (length result) :initial-contents (nreverse result)
                        :element-type 'single-float))
  (isamax n vec incx)))

(defun isamax (n dx incx)
; Tim Daly May 19, 2012
(declare (type (simple-array single-float (*)) dx)
         (type fixnum incx n))
(let ((ix 0) (dmax 0.0) (isamax -1) (limit (length dx)))
  (declare (type (single-float) dmax) (type fixnum isamax ix limit))
  (when (and (>= n 1) (> incx 0))
    (setq isamax 0)
    (setf dmax (the single-float (abs (the single-float (aref dx 0)))))
    (setf ix (the fixnum (+ ix incx)))
    (do ((i 1 (+ i 1)))
      ((or (>= ix limit) (>= i n)))
      (when (> (the single-float (abs (the single-float (aref dx ix))))) dmax)
      (setf isamax i)
      (setf dmax (the single-float (abs (the single-float (aref dx ix)))))))

```

```
(setf ix (the fixnum (+ ix incx))))
isamax))
```

— BLAS 1 isamax lisp test —

```
(load "isamax.lisp")
(setq a (vector 3.0 4.0 -3.0 5.0 -1.0))
; #(3.0 4.0 -3.0 5.0 -1.0)
(isamax 5 a 1)
; 3
(isamax 3 a 1)
; 1
(isamax 0 a 1)
; -1
(isamax -5 a 1)
; -1
(isamax 5 a -1)
; -1
(isamax 5 a 2)
; 0
(isamax 1 a 0)
; -1
(isamax 1 a -1)
; -1
(isamax 1 a 5)
; 0
(setq a (vector 3.0 4.0 -3.0 -5.0 -1.0))
; #(3.0 4.0 -3.0 -5.0 -1.0)
(isamax 5 a 1)
; 3
```

izamax BLAS

— izamax.input —

```
)set break resume
)sys rm -f izamax.output
)spool izamax.output
)set message test on
)set message auto off
)clear all
```

```
--S 1 of 6
a:PRIMARR(COMPLEX(DFLOAT)):= [[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]
--R
--R (1) [3. + 4. %i, - 4. + 5. %i, 5. + 6. %i, 7. - 8. %i, - 9. - 2. %i]
--R                                         Type: PrimitiveArray(Complex(DoubleFloat))
--E 1
```

```
--S 2 of 6
izamax(5,a,1) -- should be 3
--R
--R (2) 3
--R                                         Type: PositiveInteger
--E 2
```

```
--S 3 of 6
izamax(0,a,1) -- should be -1
--R
--R (3) - 1
--R                                         Type: Integer
--E 3
```

```
--S 4 of 6
izamax(5,a,-1) -- should be -1
--R
--R (4) - 1
--R                                         Type: Integer
--E 4
```

```
--S 5 of 6
izamax(3,a,1) -- should be 2
--R
--R (5) 2
--R                                         Type: PositiveInteger
--E 5
```

```
--S 6 of 6
izamax(3,a,2) -- should be 1
--R
--R (6) 1
--R                                         Type: PositiveInteger
--E 6
```

```
)spool
)lisp (bye)
```

— izamax.help —

```

=====
izamax examples
=====

a:PRIMARR(COMPLEX(DFLOAT)) := [[3.+4.*%i,-4.+5.*%i,5.+6.*%i,7.-8.*%i,-9.-2.*%i]]

      [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]

izamax(5,a,1)

      3

izamax(0,a,1)

      - 1

izamax(5,a,-1)

      - 1

izamax(3,a,1)

      2

izamax(3,a,2)

      1

```

```

=====
Man Page Details
=====

```

NAME

IZAMAX - BLAS level one, maximum index function

SYNOPSIS

INTEGER FUNCTION IZAMAX (n, x, incx)

INTEGER n, incx

DOUBLE COMPLEX x

DESCRIPTION

IZAMAX searches a double complex vector for the first occurrence of the maximum absolute value.

IZAMAX determines the first index i such that

$|\text{Real}(x)| + |\text{Imag}(x)| = \text{MAX}(|\text{Real}(x)| + |\text{Imag}(x)|): j = 1, \dots, n$

$\begin{matrix} i & & i & & j & & j \\ & \nearrow & & \searrow & & \nearrow & & \searrow \\ & x & & & & & & \end{matrix}$
 where x is an element of a double complex vector.
 j

ARGUMENTS

n INTEGER. (input)
 Number of elements to process in the vector to be searched. If
 $n \leq 0$, these routines return 0.

x DOUBLE COMPLEX. (input)
 Array of dimension $(n-1) * |incx| + 1$.
 Array x contains the vector to be searched.

incx INTEGER. (input)
 Increment between elements of x.

RETURN VALUES

IZAMAX INTEGER. (output)
 Return the first index of the maximum absolute value of vector
 x. The vector x has length n and increment incx.

NOTES

When working backward ($incx < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

The largest absolute value is:

$ABS(x(1+(index-1) * incx))$ when $incx > 0$

$ABS(x(1+(n-index) * |incx|))$ when $incx < 0$

— izamax.f —

```

integer function izamax(n,zx,incx)
c
c  finds the index of element having max. absolute value.
c  jack dongarra, 1/15/85.
c  modified 3/93 to return if incx .le. 0.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
double complex zx(*)
double precision smax
integer i,incx,ix,n
double precision dcabs1

```



```

c
    izamax = 0
    if( n.lt.1 .or. incx.le.0 )return
    izamax = 1
    if(n.eq.1)return
    if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
    ix = 1
    smax = dcabs1(zx(1))
    ix = ix + incx
    do 10 i = 2,n
        if(dcabs1(zx(ix)).le.smax) go to 5
        izamax = i
        smax = dcabs1(zx(ix))
    5   ix = ix + incx
    10 continue
    return
c
c      code for increment equal to 1
c
    20 smax = dcabs1(zx(1))
    do 30 i = 2,n
        if(dcabs1(zx(i)).le.smax) go to 30
        izamax = i
        smax = dcabs1(zx(i))
    30 continue
    return
end

```

— izamaxEX example —

```

program izamaxEX
*   Tim Daly May 20, 2012
*   unit tests for BLAS izamax
double complex a(5)
integer n
a = (/ (1.0D0,2.0D0), (-3.0D0,4.0D0), (5.0D0,-6.0D0),
C      (0.0D0,8.0D0), (0.0D0,-8.0D0) /)
write(6,100)dble(a(1)),dimag(a(1))
100 format("a(1)=C(",f6.3," ",f6.3,"")
write(6,101)dble(a(2)),dimag(a(2))
101 format("a(2)=C(",f6.3," ",f6.3,"")
write(6,102)dble(a(3)),dimag(a(3))
102 format("a(3)=C(",f6.3," ",f6.3,"")

```

```

        write(6,103)dbble(a(4)),dimag(a(4))
103    format("a(4)=C(",f6.3," ",f6.3,")")
        write(6,104)dbble(a(5)),dimag(a(5))
104    format("a(5)=C(",f6.3," ",f6.3,")")
        n=izamax(5,a,1)
        write(6,200)n
200    format("should be 3",/,"n=",i3)
        n=izamax(3,a,1)
        write(6,201)n
201    format("should be 3",/,"n=",i3)
        n=izamax(0,a,1)
        write(6,202)n
202    format("should be 0",/,"n=",i3)
        n=izamax(-5,a,1)
        write(6,203)n
203    format("should be 0",/,"n=",i3)
        n=izamax(5,a,-1)
        write(6,204)n
204    format("should be 0",/,"n=",i3)
        n=izamax(5,a,2)
        write(6,205)n
205    format("should be 2",/,"n=",i3)
        n=izamax(1,a,0)
        write(6,206)n
206    format("should be 0",/,"n=",i3)
        n=izamax(1,a,-1)
        write(6,207)n
207    format("should be 0",/,"n=",i3)
        n=izamax(1,a,5)
        write(6,208)n
208    format("should be 1",/,"n=",i3)
        stop
        end

```

```
gcc -o izamaxEX izamaxEX.f -lgfortran izamax.o dcabs1.o && ./izamaxEX
```

```

a(1)=C( 1.000  2.000)
a(2)=C(-3.000  4.000)
a(3)=C( 5.000 -6.000)
a(4)=C( 0.000  8.000)
a(5)=C( 0.000 -8.000)
should be 3
n= 3
should be 3
n= 3
should be 0
n= 0
should be 0
n= 0

```

```

should be 0
n= 0
should be 2
n= 2
should be 0
n= 0
should be 0
n= 0
should be 1
n= 1

```

Spad represents complex numbers as a pair where the car is the real part and the cons is the imaginary part. BLAS wants a complex number in fortran format. So we have a design choice to make. Either we could write all of the BLAS code using Spad internal representation or we could follow the BLAS code standard. I've decided to follow the BLAS code standard so we need to create thunks to do the data translation.

NOTE: The Axiom internal representation of PRIMARR(COMPLEX(DFLOAT)) is an array where each array element AE is

```
AE = (cons (therealpart 0)) (cons (theimagpart 0))
```

so we need (car AE) and (cdr AE) to get the real and imag parts.

NOTE: Array indexing in Axiom is 0-based but Fortran is 1-based so the results are displaced by 1.

— BLAS 1 izamax —

```

(defun izamaxSpad (n zx incx)
; Tim Daly May 20, 2012
  (let (result vec)
    (dotimes (i (length zx))
      (push (complex (car (svref zx i)) (cdr (svref zx i))) result))
    (setq vec (make-array (length result) :initial-contents (nreverse result)))
    (izamax n vec incx)))

(defun izamax (n zx incx)
; Tim Daly May 20, 2012
  (declare (type (simple-array (complex double-float) (*)) zx)
    (type fixnum incx n))
  (let ((ix 0) (zmax 0.0d0) (izamax -1) (limit (length zx)))
    (declare (type (double-float) zmax) (type fixnum izamax ix limit))
    (when (and (>= n 1) (> incx 0))
      (setf izamax 0)
      (setf zmax (the double-float
                    (dcabs1 (the (complex double-float) (svref zx 0)))))
      (setf ix (+ ix incx))
      (do ((i 1 (+ i 1)))

```

```

      ((or (>= ix limit) (>= i n)))
    (when (> (the double-float
              (dcabs1 (the (complex double-float) (svref zx ix))))
          zmax)
      (setf izamax i)
      (setf zmax (the double-float
                    (dcabs1 (the (complex double-float) (svref zx ix))))))
    (setf ix (the fixnum (+ ix incx))))
  izamax))

```

— BLAS 1 izamax lisp test —

```

(load "dcabs1.lisp")
(load "izamax.lisp")
(setq a (vector #C(1.0d0 2.0d0) #C(-3.0d0 4.0d0) #C(5.0d0 -6.0d0)
                #C(0.0d0 8.0d0) #C(0.0d0 -8.0d0)))
; #C(1.0 2.0) #C(-3.0 4.0) #C(5.0 -6.0) #C(0.0 8.0) #C(0.0 -8.0)
(izamax 5 a 1)
; 2
(izamax 3 a 1)
; 2
(izamax 0 a 1)
; -1
(izamax -5 a 1)
; -1
(izamax 5 a -1)
; -1
(izamax 5 a 2)
; 1
(izamax 1 a 0)
; -1
(izamax 1 a -1)
; -1
(izamax 1 a 5)
; 0

```

zaxpy BLAS

— zaxpy.input —

```

)set break resume
)sys rm -f zaxpy.output
)spool zaxpy.output
)set message test on
)set message auto off
)clear all

--S 1 of 25
a:PRIMARR(COMPLEX(DFLOAT)):=_
[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (1) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 1

--S 2 of 25
b:PRIMARR(COMPLEX(DFLOAT)):=_
[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (2) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 2

--S 3 of 25
zaxpy(3,2.0,a,1,b,1)
--R
--R (3) [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 3

--S 4 of 25
b:=[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (4) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 4

--S 5 of 25
zaxpy(5,2.0,a,1,b,1)
--R
--R (5) [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,21. - 24. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 5

--S 6 of 25
b:=[[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]
--R
--R (6) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 6

```

```

--S 7 of 25
zaxpy(3,2.0,a,3,b,3)
--R
--R (7) [9. + 12. %i,- 4. + 5. %i,5. + 6. %i,21. - 24. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 7

--S 8 of 25
b:=[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (8) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 8

--S 9 of 25
zaxpy(4,2.0,a,2,b,2)
--R
--R (9) [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 9

--S 10 of 25
b:=[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (10) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 10

--S 11 of 25
zaxpy(3,2.0,a,2,b,2)
--R
--R (11) [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 11

--S 12 of 25
b:=[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (12) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 12

--S 13 of 25
zaxpy(3,-2.0,a,1,b,2)
--R
--R (13) [- 3. - 4. %i,- 4. + 5. %i,13. - 4. %i,7. - 8. %i,- 19. - 14. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 13

```

```

--S 14 of 25
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (14) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 14

--S 15 of 25
zaxpy(3,2.0,a,1,b,2)
--R
--R (15) [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 15

--S 16 of 25
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (16) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 16

--S 17 of 25
zaxpy(-3,2.0,a,1,b,2)
--R
--R (17) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 17

--S 18 of 25
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (18) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 18

--S 19 of 25
zaxpy(3,2.0,a,-1,b,2)
--R
--R (19) [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 19

--S 20 of 25
b:=[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]
--R
--R (20) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 20

--S 21 of 25

```

```

zaxpy(3,2.0,a,1,b,-2)
--R
--R (21) [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 21

--S 22 of 25
b:=[[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (22) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 22

--S 23 of 25
zaxpy(3,2.0,a,-1,b,-2)
--R
--R (23) [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 23

--S 24 of 25
b:=[[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
--R
--R (24) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 24

--S 25 of 25
zaxpy(3,0.0,a,1,b,1)
--R
--R (25) [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
--R                                         Type: PrimitiveArray Complex DoubleFloat
--E 25

)spool
)lisp (bye)

```

— zaxpy.help —

```

=====
zaxpy examples
=====

```

```

a:PRIMARR(COMPLEX(DFLOAT)):=_
  [[3.0+4.0*%i, -4.0+5.0*%i, 5.0+6.0*%i, 7.0-8.0*%i, -9.0-2.0*%i]]
  [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]

```



```

b:PRIMARR(COMPLEX(DFLOAT)):=_
  [[3.0+4.0*i, -4.0+5.0*i, 5.0+6.0*i, 7.0-8.0*i, -9.0-2.0*i]]

  [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]

zaxpy(3,2.0,a,1,b,1)

  [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,7. - 8. %i,- 9. - 2. %i]

NOTE: We reset b to the above value before every execution

zaxpy(5,2.0,a,1,b,1)

  [9. + 12. %i,- 12. + 15. %i,15. + 18. %i,21. - 24. %i,- 27. - 6. %i]

zaxpy(3,2.0,a,3,b,3)

  [9. + 12. %i,- 4. + 5. %i,5. + 6. %i,21. - 24. %i,- 9. - 2. %i]

zaxpy(4,2.0,a,2,b,2)

  [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]

zaxpy(3,2.0,a,2,b,2)

  [9. + 12. %i,- 4. + 5. %i,15. + 18. %i,7. - 8. %i,- 27. - 6. %i]

zaxpy(3,-2.0,a,1,b,2)

  [- 3. - 4. %i,- 4. + 5. %i,13. - 4. %i,7. - 8. %i,- 19. - 14. %i]

zaxpy(3,2.0,a,1,b,2)

  [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]

zaxpy(-3,2.0,a,1,b,2)

  [3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]

zaxpy(3,2.0,a,-1,b,2)

  [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]

zaxpy(3,2.0,a,1,b,-2)

  [13. + 16. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,- 3. + 6. %i]

zaxpy(3,2.0,a,-1,b,-2)

  [9. + 12. %i,- 4. + 5. %i,- 3. + 16. %i,7. - 8. %i,1. + 10. %i]

```

```
zaxpy(3,0.0,a,1,b,1)
```

```
[3. + 4. %i,- 4. + 5. %i,5. + 6. %i,7. - 8. %i,- 9. - 2. %i]
```

```
=====
Man Page Details
=====
```

NAME

ZAXPY - BLAS level one axpy subroutine

SYNOPSIS

```
SUBROUTINE ZAXPY      ( n, alpha, x, incx, y, incy )
```

```
      INTEGER          n, incx, incy
```

```
      DOUBLE COMPLEX   alpha, x, y
```

DESCRIPTION

ZAXPY adds a scalar multiple of a double complex vector to another double complex vector.

ZAXPY computes a constant alpha times a vector x plus a vector y. The result overwrites the initial values of vector y.

This routine performs the following vector operation:

```
y <-- alpha*x + y
```

incx and incy specify the increment between two consecutive elements of respectively vector x and y.

ARGUMENTS

n INTEGER. (input)
Number of elements in the vectors. If $n \leq 0$, these routines return without any computation.

alpha DOUBLE COMPLEX. (input)
If $\alpha = 0$ this routine returns without any computation.

x DOUBLE COMPLEX. (input)
Array of dimension $(n-1) * |\text{incx}| + 1$. Contains the vector to be scaled before summation.

incx INTEGER. (input)
Increment between elements of x.
If $\text{incx} = 0$, the results will be unpredictable.

y DOUBLE COMPLEX. (input and output)
 array of dimension (n-1) * |incy| + 1.
 Before calling the routine, y contains the vector to be summed.
 After the routine ends, y contains the result of the summation.

incy INTEGER. (input)
 Increment between elements of y.
 If incy = 0, the results will be unpredictable.

NOTES

This routine is Level 1 Basic Linear Algebra Subprograms (Level 1 BLAS).

When working backward (incx < 0 or incy < 0), each routine starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)

RETURN VALUES

When n <= 0, double complex alpha = 0 = 0.+0.i, this routine returns immediately with no change in its arguments.

—————

Computes (complex double-float) $y \leftarrow \alpha x + y$

Arguments are:

- n - fixnum
- da - (complex double-float)
- dx - array (complex double-float)
- incx - fixnum
- dy - array (complex double-float)
- incy - fixnum

— zaxpy.f —

```

subroutine zaxpy(n,za,zx,incx,zy,incy)
c
c      constant times a vector plus a vector.
c      jack dongarra, 3/11/78.
c      modified 12/3/93, array(1) declarations changed to array(*)

```

```

c
double complex zx(*),zy(*),za
integer i,incx,incy,ix,iy,n
double precision dcabs1
if(n.le.0)return
if (dcabs1(za) .eq. 0.0d0) return
if (incx.eq.1.and.incy.eq.1)go to 20

c
c      code for unequal increments or equal increments
c      not equal to 1
c
ix = 1
iy = 1
if(incx.lt.0)ix = (-n+1)*incx + 1
if(incy.lt.0)iy = (-n+1)*incy + 1
do 10 i = 1,n
    zy(iy) = zy(iy) + za*zx(ix)
    ix = ix + incx
    iy = iy + incy
10 continue
return

c
c      code for both increments equal to 1
c
20 do 30 i = 1,n
    zy(i) = zy(i) + za*zx(i)
30 continue
return
end

```

— zaxpyEX example —

```

program zaxpyEX
*   Tim Daly May 20, 2012
*   unit tests for BLAS zaxpy (a*x+y)
double complex a(5)
double complex b(5)
double complex c(5)
a(1) = ( 3.0D0, 4.0D0)
a(2) = (-4.0D0, 5.0D0)
a(3) = ( 5.0D0, 6.0D0)
a(4) = ( 7.0D0,-8.0D0)
a(5) = (-9.0D0,-2.0D0)
b(1) = ( 3.0D0, 4.0D0)
b(2) = (-4.0D0, 5.0D0)
b(3) = ( 5.0D0, 6.0D0)

```

```

        b(4) = ( 7.0D0,-8.0D0)
        b(5) = (-9.0D0,-2.0D0)
        write(6,100)a(1),a(2),a(3),a(4),a(5)
100    format("a(1)=(,f7.2,,f7.2,)",/
C        "a(2)=(,f7.2,,f7.2,)",/
C        "a(3)=(,f7.2,,f7.2,)",/
C        "a(4)=(,f7.2,,f7.2,)",/
C        "a(5)=(,f7.2,,f7.2,)" )
        write(6,101)b(1),b(2),b(3),b(4),b(5)
101    format(/,"b(1)=(,f7.2,,f7.2,)",/
C        "b(2)=(,f7.2,,f7.2,)",/
C        "b(3)=(,f7.2,,f7.2,)",/
C        "b(4)=(,f7.2,,f7.2,)",/
C        "b(5)=(,f7.2,,f7.2,)" )

        call zaxpy(3,2.0d0,a,1,b,1)
        write(6,200)
200    format(/,"t200 is (9.00,12.00), (-12.00,15.00), (15.00,18.00) ",/,
C        "          (7.00,-8.00), (-9.00,-2.00)" )
        write(6,101)b(1),b(2),b(3),b(4),b(5)

        b(1) = ( 3.0D0, 4.0D0)
        b(2) = (-4.0D0, 5.0D0)
        b(3) = ( 5.0D0, 6.0D0)
        b(4) = ( 7.0D0,-8.0D0)
        b(5) = (-9.0D0,-2.0D0)
        call zaxpy(5,2.0d0,a,1,b,1)
        write(6,300)
300    format(/,"t300 is (9.00,12.00), (-12.00,15.00), (15.00,18.00) ",/,
C        "          (21.00,-24.00), (-27.00,-6.00)" )
        write(6,101)b(1),b(2),b(3),b(4),b(5)

        b(1) = ( 3.0D0, 4.0D0)
        b(2) = (-4.0D0, 5.0D0)
        b(3) = ( 5.0D0, 6.0D0)
        b(4) = ( 7.0D0,-8.0D0)
        b(5) = (-9.0D0,-2.0D0)
        call zaxpy(3,2.0d0,a,3,b,3)
        write(6,301)
301    format(/,"t301 is (9.00,12.00), (-4.00,5.00), (5.00,6.00) ",/,
C        "          (21.00,-24.00), (-9.00,-2.00)" )
        write(6,101)b(1),b(2),b(3),b(4),b(5)

        b(1) = ( 3.0D0, 4.0D0)
        b(2) = (-4.0D0, 5.0D0)
        b(3) = ( 5.0D0, 6.0D0)
        b(4) = ( 7.0D0,-8.0D0)
        b(5) = (-9.0D0,-2.0D0)
        call zaxpy(4,2.0d0,a,2,b,2)
        write(6,302)

```

```

302  format(/,"t302 is (9.00,12.00), (-4.00,5.00), (15.00,18.00) ",/,
C      "      (7.00,-8.00), (-27.00, -6.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,2.0d0,a,2,b,2)
      write(6,303)
303  format(/,"t303 is (9.00,12.00), (-4.00,5.00), (15.00,18.00) ",/,
C      "      (7.00,-8.00), (-27.00, -6.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,-2.0d0,a,1,b,2)
      write(6,304)
304  format(/,"t304 is (-3.00,-4.00), (-4.00,5.00), (13.00,-4.00) ",/,
C      "      (7.00,-8.00), (-19.00,-14.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,2.0d0,a,1,b,2)
      write(6,305)
305  format(/,"t305 is (9.00,12.00), (-4.00,5.00), (-3.00,16.00) ",/,
C      "      (7.00,-8.00), (1.00,10.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(-3,2.0d0,a,1,b,2)
      write(6,306)
306  format(/,"t306 is (3.00,4.00), (-4.00,5.00), (5.00,6.00) ",/,
C      "      (7.00,-8.00), (-9.00,-2.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)

```

```

      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,2.0d0,a,-1,b,2)
      write(6,307)
307  format(/,"t307 is (13.00,16.00), (-4.00,5.00), (-3.00,16.00) ",/,
C      "          (7.00,-8.00), (-3.00,6.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,2.0d0,a,1,b,-2)
      write(6,308)
308  format(/,"t308 is (13.00,16.00), (-4.00,5.00), (-3.00,16.00) ",/,
C      "          (7.00,-8.00), (-3.00,6.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,2.0d0,a,-1,b,-2)
      write(6,309)
309  format(/,"t309 is (9.00,12.00), (-4.00,5.00), (-3.00,16.00) ",/,
C      "          (7.00,-8.00), (1.00,10.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      b(1) = ( 3.0D0, 4.0D0)
      b(2) = (-4.0D0, 5.0D0)
      b(3) = ( 5.0D0, 6.0D0)
      b(4) = ( 7.0D0,-8.0D0)
      b(5) = (-9.0D0,-2.0D0)
      call zaxpy(3,0.0d0,a,1,b,1)
      write(6,310)
310  format(/,"t310 is (3.00,4.00), (-4.00,5.00), (5.00,6.00) ",/,
C      "          (7.00,-8.00), (-9.00,-2.00)")
      write(6,101)b(1),b(2),b(3),b(4),b(5)

      stop
      end

```

```

gcc -o zaxpyEX zaxpyEX.f -lgfortran zaxpy.o dcabs1.o && ./zaxpyEX
a(1)=( 3.00, 4.00)
a(2)=( -4.00, 5.00)

```

a(3)=(5.00, 6.00)
 a(4)=(7.00, -8.00)
 a(5)=(-9.00, -2.00)

b(1)=(3.00, 4.00)
 b(2)=(-4.00, 5.00)
 b(3)=(5.00, 6.00)
 b(4)=(7.00, -8.00)
 b(5)=(-9.00, -2.00)

t200 is (9.00,12.00), (-12.00,15.00), (15.00,18.00)
 (7.00,-8.00), (-9.00,-2.00)

b(1)=(9.00, 12.00)
 b(2)=(-12.00, 15.00)
 b(3)=(15.00, 18.00)
 b(4)=(7.00, -8.00)
 b(5)=(-9.00, -2.00)

t300 is (9.00,12.00), (-12.00,15.00), (15.00,18.00)
 (21.00,-24.00), (-27.00,-6.00)

b(1)=(9.00, 12.00)
 b(2)=(-12.00, 15.00)
 b(3)=(15.00, 18.00)
 b(4)=(21.00, -24.00)
 b(5)=(-27.00, -6.00)

t301 is (9.00,12.00), (-4.00,5.00), (5.00,6.00)
 (21.00,-24.00), (-9.00,-2.00)

b(1)=(9.00, 12.00)
 b(2)=(-4.00, 5.00)
 b(3)=(5.00, 6.00)
 b(4)=(21.00, -24.00)
 b(5)=(-9.00, -2.00)

t302 is (9.00,12.00), (-4.00,5.00), (15.00,18.00)
 (7.00,-8.00), (-27.00, -6.00)

b(1)=(9.00, 12.00)
 b(2)=(-4.00, 5.00)
 b(3)=(15.00, 18.00)
 b(4)=(7.00, -8.00)
 b(5)=(-27.00, -6.00)

t303 is (9.00,12.00), (-4.00,5.00), (15.00,18.00)
 (7.00,-8.00), (-27.00, -6.00)

b(1)=(9.00, 12.00)


```
b(2)=( -4.00,  5.00)
b(3)=( 15.00, 18.00)
b(4)=(  7.00, -8.00)
b(5)=( -27.00, -6.00)
```

```
t304 is (-3.00,-4.00), (-4.00,5.00), (13.00,-4.00)
        (7.00,-8.00), (-19.00,-14.00)
```

```
b(1)=( -3.00, -4.00)
b(2)=( -4.00,  5.00)
b(3)=( 13.00, -4.00)
b(4)=(  7.00, -8.00)
b(5)=( -19.00, -14.00)
```

```
t305 is (9.00,12.00), (-4.00,5.00), (-3.00,16.00)
        (7.00,-8.00), (1.00,10.00)
```

```
b(1)=(  9.00, 12.00)
b(2)=( -4.00,  5.00)
b(3)=( -3.00, 16.00)
b(4)=(  7.00, -8.00)
b(5)=(  1.00, 10.00)
```

```
t306 is (3.00,4.00), (-4.00,5.00), (5.00,6.00)
        (7.00,-8.00), (-9.00,-2.00)
```

```
b(1)=(  3.00,  4.00)
b(2)=( -4.00,  5.00)
b(3)=(  5.00,  6.00)
b(4)=(  7.00, -8.00)
b(5)=( -9.00, -2.00)
```

```
t307 is (13.00,16.00), (-4.00,5.00), (-3.00,16.00)
        (7.00,-8.00), (-3.00,6.00)
```

```
b(1)=( 13.00, 16.00)
b(2)=( -4.00,  5.00)
b(3)=( -3.00, 16.00)
b(4)=(  7.00, -8.00)
b(5)=( -3.00,  6.00)
```

```
t308 is (13.00,16.00), (-4.00,5.00), (-3.00,16.00)
        (7.00,-8.00), (-3.00,6.00)
```

```
b(1)=( 13.00, 16.00)
b(2)=( -4.00,  5.00)
b(3)=( -3.00, 16.00)
b(4)=(  7.00, -8.00)
b(5)=( -3.00,  6.00)
```

```
t309 is (9.00,12.00), (-4.00,5.00), (-3.00,16.00)
        (7.00,-8.00), (1.00,10.00)
```

```
b(1)=( 9.00, 12.00)
b(2)=( -4.00, 5.00)
b(3)=( -3.00, 16.00)
b(4)=( 7.00, -8.00)
b(5)=( 1.00, 10.00)
```

```
t310 is (3.00,4.00), (-4.00,5.00), (5.00,6.00)
        (7.00,-8.00), (-9.00,-2.00)
```

```
b(1)=( 3.00, 4.00)
b(2)=( -4.00, 5.00)
b(3)=( 5.00, 6.00)
b(4)=( 7.00, -8.00)
b(5)=( -9.00, -2.00)
```

Axiom delivers PRIMARR(COMPLEX(DFLOAT)) as a list of pairs. We create a thunk function `zaxpySpad` which creates an array of complex double-float objects required by `zaxpy`. The `zaxpy` function is defined to modify the second array. So the thunk function has to setf this information back into the original array.

The `zaxpySpad` function could be more efficient if it only copied elements that were modified. This was not done because of the overhead of figuring out which elements this could be under all possible arguments.

— BLAS 1 zaxpy —

```
(defun zaxpySpad (n za zx incx zy incy)
; Tim Daly May 23, 2012
  (let (result vecx vecy tx ty)
    (dotimes (i (length zx))
      (push (complex (car (svref zx i)) (cdr (svref zx i))) tx))
    (setq vecx (make-array (length tx) :initial-contents (nreverse tx)))
    (dotimes (i (length zy))
      (push (complex (car (svref zy i)) (cdr (svref zy i))) ty))
    (setq vecy (make-array (length ty) :initial-contents (nreverse ty)))
    (zaxpy n (complex (car za) (cdr za)) vecx incx vecy incy)
    (dotimes (i (length vecx))
      (setf (svref zy i)
        (cons (realpart (svref vecy i)) (imagpart (svref vecy i)))))
    zy))

(defun zaxpy (n za zx incx zy incy)
; Tim Daly May 23, 2012
  (declare (type (simple-array (complex double-float) (*)) zy zx)
    (type (complex double-float) za)
    (type fixnum incy incx n))
```

```

(let ((ix 0) (iy 0) (limitx (length zx)) (limity (length zy)))
  (declare (type fixnum iy ix limitx limity))
  (when (and (> n 0) (/= (dcabs1 za) 0.0))
    (when (< incx 0)
      (setf ix (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incx))))
    (when (< incy 0)
      (setf iy (the fixnum (* (the fixnum (1+ (the fixnum (- n)))) incy))))
    (do ((i 0 (1+ i)))
        ((or (>= i n) (< ix 0) (< iy 0) (>= ix limitx) (>= iy limity)))
      (setf (the (complex double-float) (svref zy iy))
        (+ (the (complex double-float) (svref zy iy))
          (the (complex double-float)
            (* za (the (complex double-float) (svref zx ix))))))
      (setf ix (+ ix incx))
      (setf iy (+ iy incy)))))

```

— BLAS 1 izamax lisp test —

```

(load "zaxpy.lisp")
(load "dcabs1.lisp")
(setq zx (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx 1 zy 1) ; t200
; #((9.0 . 12.0) (-12.0 . 15.0) (15.0 . 18.0) (7.0 . -8.0) (-9.0 . -2.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 5 2.0d0 zx 1 zy 1) ; t300
; #((9.0 . 12.0) (-12.0 . 15.0) (15.0 . 18.0) (21.0 . -24.0) (-27.0 . -6.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx 3 zy 3) ; t301
; #((9.0 . 12.0) (-4.0 . 5.0) (5.0 . 6.0) (21.0 . -24.0) (-9.0 . -2.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 4 2.0d0 zx 2 zy 2) ; t302
; #((9.0 . 12.0) (-4.0 . 5.0) (15.0 . 18.0) (7.0 . -8.0) (-27.0 . -6.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))

```

```

(zaxpyspad 3 2.0d0 zx 2 zy 2) ; t303
; #((9.0 . 12.0) (-4.0 . 5.0) (15.0 . 18.0) (7.0 . -8.0) (-27.0 . -6.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 -2.0d0 zx 1 zy 2) ; t304
; #((-3.0 . -4.0) (-4.0 . 5.0) (13.0 . -4.0) (7.0 . -8.0) (-19.0 . -14.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx 1 zy 2) ; t305
; #((9.0 . 12.0) (-4.0 . 5.0) (-3.0 . 16.0) (7.0 . -8.0) (1.0 . 10.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad -3 2.0d0 zx 1 zy 2) ; t306
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx -1 zy 2) ; t307
; #((13.0 . 16.0) (-4.0 . 5.0) (-3.0 . 16.0) (7.0 . -8.0) (-3.0 . 6.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx 1 zy -2) ; t308
; #((13.0 . 16.0) (-4.0 . 5.0) (-3.0 . 16.0) (7.0 . -8.0) (-3.0 . 6.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 2.0d0 zx -1 zy -2) ; t309
; #((9.0 . 12.0) (-4.0 . 5.0) (-3.0 . 16.0) (7.0 . -8.0) (1.0 . 10.0))
(setq zy (vector (cons 3.0d0 4.0d0) (cons -4.0d0 5.0d0) (cons 5.0d0 6.0d0)
                 (cons 7.0d0 -8.0d0) (cons -9.0d0 -2.0d0)))
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))
(zaxpyspad 3 0.0d0 zx 1 zy 1) ; t310
; #((3.0 . 4.0) (-4.0 . 5.0) (5.0 . 6.0) (7.0 . -8.0) (-9.0 . -2.0))

```

zcopy BLAS

— zcopy.input —

```

)set break resume
)sys rm -f zcopy.output

```

```

)spool zcopy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zcopy.help —

```

=====
zcopy examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZCOPY - BLAS level one, copies a double complex vector into another double complex vector

SYNOPSIS

```
SUBROUTINE ZCOPY    ( n, x, incx, y, incy )
```

```
    INTEGER          n, incx, incy
```

```
    DOUBLE COMPLEX   x, y
```

DESCRIPTION

ZCOPY copies a double complex vector into another double complex vector. ZCOPY copies a vector x, whose length is n to a vector y. incx and incy specify the increment between two consecutive elements of respectively vector x and y.

This routine performs the following vector operation:

```
y <-- x
```

where x and y are double complex vectors.

ARGUMENTS

```

n          INTEGER. (input)
           Number of vector elements to be copied.
           If n <= 0, this routine returns without computation.

```

x DOUBLE COMPLEX, (input)
 Vector from which to copy.

incx INTEGER. (input)
 Increment between elements of x.
 If incx = 0, the results will be unpredictable.

y DOUBLE COMPLEX, (output)
 array of dimension (n-1) * |incy| + 1, result vector.

incy INTEGER. (input)
 Increment between elements of y. If incy = 0, the results will
 be unpredictable.

NOTES

When working backward (incx < 0 or incy < 0), each routine starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)

—————

— zcopy.f —

```

subroutine zcopy(n,zx,incx,zy,incy)
c
c  copies a vector, x, to a vector, y.
c  jack dongarra, linpack, 4/11/78.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
  double complex zx(*),zy(*)
  integer i,incx,incy,ix,iy,n
c
  if(n.le.0)return
  if(incx.eq.1.and.incy.eq.1)go to 20
c
c      code for unequal increments or equal increments
c      not equal to 1
c
  ix = 1
  iy = 1
  if(incx.lt.0)ix = (-n+1)*incx + 1
  if(incy.lt.0)iy = (-n+1)*incy + 1
  do 10 i = 1,n
    zy(iy) = zx(ix)
    ix = ix + incx

```

```

        iy = iy + incy
10 continue
    return
c
c        code for both increments equal to 1
c
20 do 30 i = 1,n
    zy(i) = zx(i)
30 continue
    return
end

```

— BLAS 1 zcopy —

```

(defun zcopy (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
    (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
    ((zx (complex double-float) zx-%data% zx-%offset%)
     (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0))
      (declare (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
        (setf ix
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incx)
            1)))
      (if (< incy 0)
        (setf iy
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incy)
            1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%)
            (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
    label20
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))

```

```

                                (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%)
              (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)))
      end_label
      (return (values nil nil nil nil nil))))

```

zdotc BLAS

— zdotc.input —

```

)set break resume
)sys rm -f zdotc.output
)spool zdotc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zdotc.help —

```

=====
zdotc examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZDOTC - BLAS level one, computes the hermitian dot product of vector x and vector y.

SYNOPSIS

DOUBLE COMPLEX FUNCTION ZDOTC (n, x, incx, y, incy)

INTEGER n, incx, incy

COMPLEX*16 x, y

DESCRIPTION

ZDOTC computes a dot product of the conjugate of a complex vector and another complex vector (1 complex inner product).
2

ZDOTC computes a dot product of the conjugate of a complex vector and another complex vector (1 complex inner product).
2

This routine performs the following vector operation:

$$\begin{aligned} \text{ZDOTC} &\leftarrow (\text{conjugate transpose of } x) * y \\ &= \sum_{i=1}^n (\text{complex conjugate of } x(i)) * y(i) \end{aligned}$$

H

where x and y are complex vectors, and x is the conjugate transpose of x .

If $n \leq 0$, ZDOTC is set to 0.

ARGUMENTS

n INTEGER. (input)
 Number of elements in each vector.

x COMPLEX*16. (input)
 Array of dimension $(n-1) * |incx| + 1$.
 Array x contains the first vector operand.

$incx$ INTEGER. (input)
 Increment between elements of x .
 If $incx = 0$, the results will be unpredictable.

y COMPLEX*16. (input)
 Array of dimension $(n-1) * |incy| + 1$.
 Array y contains the second vector operand.

$incy$ INTEGER. (input)
 Increment between elements of y .
 If $incy = 0$, the results will be unpredictable.

RETURN VALUES

ZDOTC DOUBLE COMPLEX. Result (dot product). (output)

NOTES

When working backward ($incx < 0$ or $incy < 0$), each routine starts at the end of the vector and moves backward, as follows:

x(1-incx * (n-1)), x(1-incx * (n-2)), ..., x(1)

y(1-incy * (n-1)), y(1-incy * (n-2)), ..., y(1)

— zdotc.f —

```

double complex function zdotc(n,zx,incx,zy,incy)
c
c  forms the dot product of a vector.
c  jack dongarra, 3/11/78.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
double complex zx(*),zy(*),ztemp
integer i,incx,incy,ix,iy,n
ztemp = (0.0d0,0.0d0)
zdotc = (0.0d0,0.0d0)
if(n.le.0)return
if(incx.eq.1.and.incy.eq.1)go to 20
c
c      code for unequal increments or equal increments
c      not equal to 1
c
ix = 1
iy = 1
if(incx.lt.0)ix = (-n+1)*incx + 1
if(incy.lt.0)iy = (-n+1)*incy + 1
do 10 i = 1,n
    ztemp = ztemp + dconjg(zx(ix))*zy(iy)
    ix = ix + incx
    iy = iy + incy
10 continue
zdotc = ztemp
return
c
c      code for both increments equal to 1
c
20 do 30 i = 1,n
    ztemp = ztemp + dconjg(zx(i))*zy(i)
30 continue
zdotc = ztemp
return
end

```

— BLAS 1 zdotc —

```

(defun zdotc (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
    (type fixnum incx incy n))
  (f2cl-lib:with-multi-array-data
    ((zx (complex double-float) zx-%data% zx-%offset%)
     (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)) (zdotc #C(0.0 0.0)))
      (declare (type (complex double-float) zdotc ztemp)
        (type fixnum iy ix i))
      (setf ztemp (complex 0.0 0.0))
      (setf zdotc (complex 0.0 0.0))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
        (setf ix
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incx)
            1)))
      (if (< incy 0)
        (setf iy
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incy)
            1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf ztemp
            (+ ztemp
              (*
                (f2cl-lib:dconjg
                  (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
                (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))))
            (setf ix (f2cl-lib:int-add ix incx))
            (setf iy (f2cl-lib:int-add iy incy))))
      (setf zdotc ztemp)
      (go end_label)
    label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf ztemp
            (+ ztemp
              (*
                (f2cl-lib:dconjg
                  (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))

```

```

                                (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))))))
      (setf zdotc ztemp)
end_label
      (return (values zdotc nil nil nil nil nil))))

```

zdotu BLAS

— zdotu.input —

```

)set break resume
)sys rm -f zdotu.output
)spool zdotu.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zdotu.help —

```

=====
zdotu examples
=====

=====
Man Page Details
=====

NAME
    ZDOTU - BLAS level one, computes computes a dot product (inner product)
    of two complex vectors

SYNOPSIS
    DOUBLE COMPLEX FUNCTION ZDOTU ( n, x, incx, y, incy )

    INTEGER                                n, incx, incy

    COMPLEX*16                             x, y

```

DESCRIPTION

ZDOTU computes a dot product of two complex vectors.

This routine performs the following vector operation:

$$\text{ZDOTU} \leftarrow (\text{transpose of } x) * y = \sum_{i=1}^n x(i)^T * y(i)$$

where x and y are real vectors, and x is the transpose of x .

If $n \leq 0$, ZDOTU is set to 0.

ARGUMENTS

n INTEGER. (input)
Number of elements in each vector.

x COMPLEX*16. (input)
Array of dimension $(n-1) * |incx| + 1$.
Array x contains the first vector operand.

$incx$ INTEGER. (input)
Increment between elements of x .
If $incx = 0$, the results will be unpredictable.

y COMPLEX*16. (input)
Array of dimension $(n-1) * |incy| + 1$.
Array y contains the second vector operand.

$incy$ INTEGER. (input)
Increment between elements of y .
If $incy = 0$, the results will be unpredictable.

RETURN VALUES

ZDOTU DOUBLE COMPLEX. Result (dot product). (output)

NOTES

When working backward ($incx < 0$ or $incy < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-incx * (n-1)), x(1-incx * (n-2)), \dots, x(1)$

$y(1-incy * (n-1)), y(1-incy * (n-2)), \dots, y(1)$

— zdotu.f —

```

double complex function zdotu(n,zx,incx,zy,incy)
c
c   forms the dot product of two vectors.
c   jack dongarra, 3/11/78.
c   modified 12/3/93, array(1) declarations changed to array(*)
c
double complex zx(*),zy(*),ztemp
integer i,incx,incy,ix,iy,n
ztemp = (0.0d0,0.0d0)
zdotu = (0.0d0,0.0d0)
if(n.le.0)return
if(incx.eq.1.and.incy.eq.1)go to 20
c
c   code for unequal increments or equal increments
c   not equal to 1
c
ix = 1
iy = 1
if(incx.lt.0)ix = (-n+1)*incx + 1
if(incy.lt.0)iy = (-n+1)*incy + 1
do 10 i = 1,n
    ztemp = ztemp + zx(ix)*zy(iy)
    ix = ix + incx
    iy = iy + incy
10 continue
zdotu = ztemp
return
c
c   code for both increments equal to 1
c
20 do 30 i = 1,n
    ztemp = ztemp + zx(i)*zy(i)
30 continue
zdotu = ztemp
return
end

```

— BLAS 1 zdotu —

```

(defun zdotu (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
    (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
    ((zx (complex double-float) zx-%data% zx-%offset%)
     (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)) (zdotu #C(0.0 0.0)))

```

```

(declare (type (complex double-float) zdotu ztemp)
  (type fixnum iy ix i))
(setf ztemp (complex 0.0 0.0))
(setf zdotu (complex 0.0 0.0))
(if (<= n 0) (go end_label))
(if (and (= incx 1) (= incy 1)) (go label20))
(setf ix 1)
(setf iy 1)
(if (< incx 0)
  (setf ix
    (f2cl-lib:int-add
      (f2cl-lib:int-mul (the fixnum (1- n)) incx)
      1)))
(if (< incy 0)
  (setf iy
    (f2cl-lib:int-add
      (f2cl-lib:int-mul (the fixnum (1- n)) incy)
      1)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i n) nil)
  (tagbody
    (setf ztemp
      (+ ztemp
        (* (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
          (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))))
    (setf ix (f2cl-lib:int-add ix incx))
    (setf iy (f2cl-lib:int-add iy incy))))
(setf zdotu ztemp)
(go end_label)
label20
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i n) nil)
  (tagbody
    (setf ztemp
      (+ ztemp
        (* (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
          (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))))
    (setf zdotu ztemp)
  end_label
  (return (values zdotu nil nil nil nil nil))))

```

zdscal BLAS

— zdscal.input —

```

)set break resume
)sys rm -f zdscal.output
)spool zdscal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zdscal.help —

```

=====
zdscal examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZDSCAL - BLAS level one, Scales a double complex vector

SYNOPSIS

```

SUBROUTINE ZDSCAL    ( n, alpha, x, incx )

```

```

    INTEGER          n, incx

```

```

    DOUBLE COMPLEX    x

```

```

    DOUBLE PRECISION  alpha

```

DESCRIPTION

ZDSCAL scales a double complex vector with a double precision scalar. ZDSCAL scales the vector x of length n and increment incx by the constant alpha.

This routine performs the following vector operation:

$$x \leftarrow \alpha x$$

where alpha is a double precision scalar, and x is a double complex vector.

ARGUMENTS

```

n          INTEGER. (input)

```


Number of elements in the vector.
 If $n \leq 0$, this routine returns without computation.

alpha DOUBLE PRECISION. (input)
 Value used to scale vector

x DOUBLE COMPLEX. (input and output)
 Array of dimension $(n-1) * \text{abs}(\text{incx}) + 1$. Vector to be scaled.

incx INTEGER. (input)
 Increment between elements of x.
 If $\text{incx} = 0$, the results will be unpredictable.

NOTES

When working backward ($\text{incx} < 0$ or $\text{incy} < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-\text{incx} * (n-1)), x(1-\text{incx} * (n-2)), \dots, x(1)$

— zdscal.f —

```

subroutine zdscal(n,da,zx,incx)
c
c  scales a vector by a constant.
c  jack dongarra, 3/11/78.
c  modified 3/93 to return if incx .le. 0.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
c  double complex zx(*)
c  double precision da
c  integer i,incx,ix,n
c
c  if( n.le.0 .or. incx.le.0 )return
c  if(incx.eq.1)go to 20
c
c      code for increment not equal to 1
c
c      ix = 1
c      do 10 i = 1,n
c          zx(ix) = dcmplx(da,0.0d0)*zx(ix)
c          ix = ix + incx
c  10 continue
c      return
c
c      code for increment equal to 1
c

```

```

20 do 30 i = 1,n
    zx(i) = dcplx(da,0.0d0)*zx(i)
30 continue
return
end

```

— BLAS 1 zdscal —

```

(defun zdscal (n da zx incx)
  (declare (type (simple-array (complex double-float) (*)) zx)
           (type (double-float) da)
           (type fixnum incx n))
  (f2cl-lib:with-multi-array-data
    ((zx (complex double-float) zx-%data% zx-%offset%))
    (prog ((i 0) (ix 0))
      (declare (type fixnum ix i))
      (if (or (<= n 0) (<= incx 0)) (go end_label))
      (if (= incx 1) (go label20))
      (setf ix 1)
      (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
                    (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
              (* (coerce (complex da 0.0) '(complex doublefloat))
                 (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)))
        (setf ix (f2cl-lib:int-add ix incx))))
      (go end_label)
    label20
      (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
                    (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
              (* (coerce (complex da 0.0) '(complex double-float))
                 (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))))
      end_label
      (return (values nil nil nil nil))))

```

zrotg BLAS

— zrotg.input —

```

)set break resume
)sys rm -f zrotg.output
)spool zrotg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zrotg.help —

```

=====
zrotg examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZROTG - Extensions to BLAS level one rotation subroutines

SYNOPSIS

SUBROUTINE ZROTG (a, b, c, s)

DOUBLE COMPLEX

a, b, s

DOUBLE PRECISION

c

DESCRIPTION

ZROTG computes the elements of a Givens plane rotation matrix such that:

$$\begin{bmatrix} c & s \\ -\text{conj}(s) & c \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

where $r = (a / \sqrt{\text{conj}(a)*a}) * \sqrt{\text{conj}(a)*a + \text{conj}(b)*b}$,
and the notation $\text{conj}(z)$ represents the complex conjugate of z .

The Givens plane rotation can be used to introduce zero elements into a matrix selectively.

ARGUMENTS

a (input and output) DOUBLE COMPLEX

First vector component.

On input, the first component of the vector to be rotated. On output, a is overwritten by the unique complex number r, whose size in the complex plane is the Euclidean norm of the complex vector (a,b), and whose direction in the complex plane is the same as that of the original complex element a.

```

if |a| != 0
r = a / |a| * sqrt( conjg(a)*a + conjg(b)*b )

if |a| = 0
r = b

```

b (input) DOUBLE COMPLEX

Second vector component.

The second component of the vector to be rotated.

c (output) DOUBLE PRECISION

Cosine of the angle of rotation.

```

if |a| != 0
c = |a| / sqrt( conjg(a)*a + conjg(b)*b )

if |a| = 0
c = 0

```

s (output) DOUBLE COMPLEX

Sine of the angle of rotation.

```

if |a| != 0
c=a/|a|*conjg(b)/sqrt(conjg(a)*a+conjg(b)*b)

if |a| = 0
s = ( 1.0 , 0.0 )

```

(Complex Double-Float). Computes plane rotation. Arguments are:

- da - (complex double-float)
- db - (complex double-float)
- c - double-float
- s - (complex double-float)

Returns multiple values where:

- 1 da - ca
- 2 db - nil
- 3 c - c
- 4 s - s

— **zrotg.f** —

```

subroutine zrotg(ca,cb,c,s)
double complex ca,cb,s
double precision c
double precision norm,scale
double complex alpha
if (cdabs(ca) .ne. 0.0d0) go to 10
  c = 0.0d0
  s = (1.0d0,0.0d0)
  ca = cb
  go to 20
10 continue
  scale = cdabs(ca) + cdabs(cb)
  norm = scale*dsqrt((cdabs(ca/dcmplx(scale,0.0d0))**2 +
*                (cdabs(cb/dcmplx(scale,0.0d0))**2)
  alpha = ca /cdabs(ca)
  c = cdabs(ca) / norm
  s = alpha * dconjg(cb) / norm
  ca = alpha * norm
20 continue
  return
end

```

— **BLAS 1 zrotg** —

```

(defun zrotg (ca cb c s)
  (declare (type (double-float) c) (type (complex double-float) s cb ca))
  (prog ((alpha #C(0.0 0.0)) (norm 0.0) (scale 0.0))
    (declare (type (double-float) scale norm)
              (type (complex double-float) alpha))
    (if (/= (f2cl-lib:cdabs ca) 0.0) (go label10))
    (setf c 0.0)
    (setf s (complex 1.0 0.0))
    (setf ca cb)
  (go label20)
label10
  (setf scale
    (coerce (+ (f2cl-lib:cdabs ca) (f2cl-lib:cdabs cb)) 'double-float))
  (setf norm
    (* scale
      (f2cl-lib:dsqrt
        (+ (expt (f2cl-lib:cdabs (/ ca
          (coerce (complex scale 0.0) '(complex double-float)))) 2)
          (expt (f2cl-lib:cdabs (/ cb
            (coerce (complex scale 0.0) '(complex double-float))))
            2))))))
    (setf alpha (/ ca (f2cl-lib:cdabs ca)))
    (setf c (/ (f2cl-lib:cdabs ca) norm))
    (setf s (/ (* alpha (f2cl-lib:dconjg cb)) norm))
    (setf ca (* alpha norm))
  label20
    (return (values ca nil c s))))

```

zscal BLAS

— zscal.input —

```

)set break resume
)sys rm -f zscal.output
)spool zscal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zscal.help —

```
=====
zscal examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZSCAL - BLAS level one, Scales a double complex vector

SYNOPSIS

```
SUBROUTINE ZSCAL    ( n, alpha, x, incx )
```

```
    INTEGER          n, incx
```

```
    DOUBLE COMPLEX   x, alpha
```

DESCRIPTION

ZSCAL scales a double complex vector with a double complex scalar. ZSCAL scales the vector x of length n and increment incx by the constant alpha.

This routine performs the following vector operation:

$$x \leftarrow \alpha x$$

where alpha is a double complex scalar, and x is a double complex vector.

ARGUMENTS

| | |
|-------|---|
| n | INTEGER. (input) Number of elements in the vector. If n <= 0, this routine returns without computation. |
| alpha | DOUBLE COMPLEX. (input) Value used to scale vector |
| x | DOUBLE COMPLEX. (input and output) Array of dimension (n-1) * abs(incx) + 1. Vector to be scaled. |
| incx | INTEGER. (input) Increment between elements of x. If incx = 0, the results will be unpredictable. |

NOTES

When working backward ($\text{incx} < 0$ or $\text{incy} < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-\text{incx} * (n-1)), x(1-\text{incx} * (n-2)), \dots, x(1)$

— zscal.f —

```

subroutine zscal(n,za,zx,incx)
c
c  scales a vector by a constant.
c  jack dongarra, 3/11/78.
c  modified 3/93 to return if incx .le. 0.
c  modified 12/3/93, array(1) declarations changed to array(*)
c
  double complex za,zx(*)
  integer i,incx,ix,n
c
  if( n.le.0 .or. incx.le.0 )return
  if(incx.eq.1)go to 20
c
  code for increment not equal to 1
c
  ix = 1
  do 10 i = 1,n
    zx(ix) = za*zx(ix)
    ix = ix + incx
  10 continue
  return
c
  code for increment equal to 1
c
  20 do 30 i = 1,n
    zx(i) = za*zx(i)
  30 continue
  return
end

```

— BLAS 1 zscal —

```

(defun zscal (n za zx incx)
  (declare (type (simple-array (complex double-float) (*)) zx)
           (type (complex double-float) za)

```



```

        (type fixnum incx n))
(f2cl-lib:with-multi-array-data
  ((zx (complex double-float) zx-%data% zx-%offset%))
  (prog ((i 0) (ix 0))
    (declare (type fixnum ix i))
    (if (or (<= n 0) (<= incx 0)) (go end_label))
    (if (= incx 1) (go label20))
    (setf ix 1)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
          (* za (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)))
        (setf ix (f2cl-lib:int-add ix incx))))
    (go end_label)
  label20
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
          (* za (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))))))
  end_label
  (return (values nil nil nil nil))))

```

zswap BLAS

— zswap.input —

```

)set break resume
)sys rm -f zswap.output
)spool zswap.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zswap.help —

=====

zswap examples

Man Page Details

NAME

ZSWAP - BLAS level one, Swaps two double complex vectors

SYNOPSIS

SUBROUTINE ZSWAP (n, x, incx, y, incy)

INTEGER n, incx, incy

DOUBLE COMPLEX x, y

DESCRIPTION

ZSWAP swaps two double complex vectors, it interchanges n values of vector x and vector y. incx and incy specify the increment between two consecutive elements of respectively vector x and y.

This routine performs the following vector operation:

$$x \leftrightarrow y$$

where x and y are double complex vectors.

ARGUMENTS

n INTEGER. (input)
Number of vector elements to be swapped.
If n <= 0, this routine returns without computation.

x DOUBLE COMPLEX, (input and output)
Array of dimension (n-1) * |incx| + 1.

incx INTEGER. (input)
Increment between elements of x.
If incx = 0, the results will be unpredictable.

y DOUBLE COMPLEX, (input and output)
array of dimension (n-1) * |incy| + 1. Vector to be swapped.

incy INTEGER. (input)
Increment between elements of y. If incy = 0, the results will be unpredictable.

NOTES

When working backward ($\text{incx} < 0$ or $\text{incy} < 0$), each routine starts at the end of the vector and moves backward, as follows:

$x(1-\text{incx} * (n-1)), x(1-\text{incx} * (n-2)), \dots, x(1)$

$y(1-\text{incy} * (n-1)), y(1-\text{incy} * (n-2)), \dots, y(1)$

—

— zswap.f —

```

subroutine zswap (n,zx,incx,zy,incy)
c
c   interchanges two vectors.
c   jack dongarra, 3/11/78.
c   modified 12/3/93, array(1) declarations changed to array(*)
c
c   double complex zx(*),zy(*),ztemp
c   integer i,incx,incy,ix,iy,n
c
c   if(n.le.0)return
c   if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments not equal
c       to 1
c
c       ix = 1
c       iy = 1
c       if(incx.lt.0)ix = (-n+1)*incx + 1
c       if(incy.lt.0)iy = (-n+1)*incy + 1
c       do 10 i = 1,n
c           ztemp = zx(ix)
c           zx(ix) = zy(iy)
c           zy(iy) = ztemp
c           ix = ix + incx
c           iy = iy + incy
10  continue
c       return
c
c       code for both increments equal to 1
20  do 30 i = 1,n
c           ztemp = zx(i)
c           zx(i) = zy(i)
c           zy(i) = ztemp
30  continue
c       return
c       end

```

— BLAS 1 zswap —

```

(defun zswap (n zx incx zy incy)
  (declare (type (simple-array (complex double-float) (*)) zy zx)
    (type fixnum incy incx n))
  (f2cl-lib:with-multi-array-data
    ((zx (complex double-float) zx-%data% zx-%offset%)
     (zy (complex double-float) zy-%data% zy-%offset%))
    (prog ((i 0) (ix 0) (iy 0) (ztemp #C(0.0 0.0)))
      (declare (type (complex double-float) ztemp)
        (type fixnum iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1)
      (setf iy 1)
      (if (< incx 0)
        (setf ix
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incx)
            1)))
      (if (< incy 0)
        (setf iy
          (f2cl-lib:int-add
            (f2cl-lib:int-mul (the fixnum (1- n)) incy)
            1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf ztemp (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%))
          (setf (f2cl-lib:fref zx-%data% (ix) ((1 *)) zx-%offset%)
            (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%))
          (setf (f2cl-lib:fref zy-%data% (iy) ((1 *)) zy-%offset%) ztemp)
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))))
      (go end_label)
    label20
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i n) nil)
      (tagbody
        (setf ztemp (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%))
        (setf (f2cl-lib:fref zx-%data% (i) ((1 *)) zx-%offset%)
          (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%))
        (setf (f2cl-lib:fref zy-%data% (i) ((1 *)) zy-%offset%) ztemp)))
    end_label
    (return (values nil nil nil nil nil))))

```

Chapter 4

BLAS Level 2

dgbmv BLAS

— dgbmv.input —

```
)set break resume
)sys rm -f dgbmv.output
)spool dgbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dgbmv.help —

```
=====
dgbmv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGBMV - perform one of the matrix-vector operations $y := \alpha A x + \beta y$, or $y := \alpha A' x + \beta y$,

SYNOPSIS

```
SUBROUTINE DGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X,
                  INCX, BETA, Y, INCY )
```

```
DOUBLE          PRECISION ALPHA, BETA
```

```
INTEGER         INCX, INCY, KL, KU, LDA, M, N
```

```
CHARACTER*1     TRANS
```

```
DOUBLE          PRECISION A( LDA, * ), X( * ), Y( * )
```

PURPOSE

DGBMV performs one of the matrix-vector operations

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

PARAMETERS

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' y := alpha*A*x + beta*y.

TRANS = 'T' or 't' y := alpha*A'*x + beta*y.

TRANS = 'C' or 'c' y := alpha*A'*x + beta*y.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL - INTEGER.

On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy 0 .le. KL. Unchanged on exit.

KU - INTEGER.

On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy 0 .le. KU.

Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).
Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N K = KU + 1 - J DO 10, I = MAX( 1, J -
KU ), MIN( M, J + KL ) A( K + I, J ) = matrix( I, J )
10    CONTINUE 20 CONTINUE
```

Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1). Unchanged on exit.

X - DOUBLE PRECISION array of DIMENSION at least
(1 + (n - 1) * abs(INCX)) when TRANS = 'N' or 'n'
and at least (1 + (m - 1) * abs(INCX)) otherwise.
Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y - DOUBLE PRECISION array of DIMENSION at least
(1 + (m - 1) * abs(INCY)) when TRANS = 'N' or 'n'

and at least $(1 + (n - 1) * \text{abs}(\text{ INCY }))$ otherwise.
 Before entry, the incremented array Y must contain
 the vector y. On exit, Y is overwritten by the
 updated vector y.

INCY - INTEGER.

On entry, INCY specifies the increment for the ele-
 ments of Y. INCY must not be zero. Unchanged on
 exit.

— dgbmv.f —

```

SUBROUTINE DGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA, BETA
INTEGER           INCX, INCY, KL, KU, LDA, M, N
CHARACTER*1       TRANS
*
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
*
* ..
*
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
* .. Parameters ..
DOUBLE PRECISION  ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
* .. Local Scalars ..
DOUBLE PRECISION  TEMP
INTEGER           I, INFO, IX, IY, J, JX, JY, K, KUP1, KX, KY,
$                LENX, LENY
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC         MAX, MIN
*
* ..

```

```

*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( TRANS, 'N' ) .AND.
$           .NOT.LSAME( TRANS, 'T' ) .AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 1
      ELSE IF( M.LT.0 ) THEN
          INFO = 2
      ELSE IF( N.LT.0 ) THEN
          INFO = 3
      ELSE IF( KL.LT.0 ) THEN
          INFO = 4
      ELSE IF( KU.LT.0 ) THEN
          INFO = 5
      ELSE IF( LDA.LT.( KL + KU + 1 ) ) THEN
          INFO = 8
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 10
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 13
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DGBMV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( ( M.EQ.0 ) .OR. ( N.EQ.0 ) .OR.
$      ( ( ALPHA.EQ.ZERO ) .AND. ( BETA.EQ.ONE ) ) )
$      RETURN
*
*      Set LENX and LENY, the lengths of the vectors x and y, and set
*      up the start points in X and Y.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
          LENX = N
          LENY = M
      ELSE
          LENX = M
          LENY = N
      END IF
      IF( INCX.GT.0 ) THEN
          KX = 1
      ELSE
          KX = 1 - ( LENX - 1 ) * INCX
      END IF

```

```

      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( LENY - 1 )*INCY
      END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through the band part of A.
*
*   First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, LENY
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, LENY
              Y( I ) = BETA*Y( I )
20          CONTINUE
            END IF
          ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
              DO 30, I = 1, LENY
                Y( IY ) = ZERO
                IY = IY + INCY
30          CONTINUE
              ELSE
                DO 40, I = 1, LENY
                  Y( IY ) = BETA*Y( IY )
                  IY = IY + INCY
40          CONTINUE
                END IF
              END IF
            END IF
            IF( ALPHA.EQ.ZERO )
$      RETURN
            KUP1 = KU + 1
            IF( LSAME( TRANS, 'N' ) )THEN
*
*   Form y := alpha*A*x + y.
*
*
              JX = KX
              IF( INCY.EQ.1 )THEN
                DO 60, J = 1, N
                  IF( X( JX ).NE.ZERO )THEN
                    TEMP = ALPHA*X( JX )
                    K = KUP1 - J

```

```

DO 50, I = MAX( 1, J - KU ), MIN( M, J + KL )
    Y( I ) = Y( I ) + TEMP*A( K + I, J )
50    CONTINUE
    END IF
    JX = JX + INCX
60    CONTINUE
ELSE
    DO 80, J = 1, N
        IF( X( JX ).NE.ZERO )THEN
            TEMP = ALPHA*X( JX )
            IY = KY
            K = KUP1 - J
            DO 70, I = MAX( 1, J - KU ), MIN( M, J + KL )
                Y( IY ) = Y( IY ) + TEMP*A( K + I, J )
                IY = IY + INCY
70            CONTINUE
            END IF
            JX = JX + INCX
            IF( J.GT.KU )
                $    KY = KY + INCY
80            CONTINUE
        END IF
    ELSE
*
*       Form y := alpha*A'*x + y.
*
        JY = KY
        IF( INCX.EQ.1 )THEN
            DO 100, J = 1, N
                TEMP = ZERO
                K = KUP1 - J
                DO 90, I = MAX( 1, J - KU ), MIN( M, J + KL )
                    TEMP = TEMP + A( K + I, J )*X( I )
90                CONTINUE
                Y( JY ) = Y( JY ) + ALPHA*TEMP
                JY = JY + INCY
100            CONTINUE
        ELSE
            DO 120, J = 1, N
                TEMP = ZERO
                IX = KX
                K = KUP1 - J
                DO 110, I = MAX( 1, J - KU ), MIN( M, J + KL )
                    TEMP = TEMP + A( K + I, J )*X( IX )
                    IX = IX + INCX
110            CONTINUE
                Y( JY ) = Y( JY ) + ALPHA*TEMP
                JY = JY + INCY
                IF( J.GT.KU )
                    $    KX = KX + INCX

```

```

120      CONTINUE
      END IF
    END IF
*
      RETURN
*
*   End of DGBMV .
*
      END

```

— BLAS 2 dgbmv —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dgbmv (trans m n kl ku alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
              (type (double-float) beta alpha)
              (type fixnum incx lda ku kl n m)
              (type character trans))
    (f2cl-lib:with-multi-array-data
      ((trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kup1 0)
              (kx 0) (ky 0) (lenx 0) (leny 0) (temp 0.0))
        (declare (type fixnum i info ix iy j jx jy k kup1 kx ky
                              lenx leny)
                  (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                 (not (char-equal trans #\T))
                 (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< kl 0)
           (setf info 4))
          ((< ku 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add kl ku 1))
           (setf info 8))

```

```

      ((= incx 0)
       (setf info 10))
      ((= incy 0)
       (setf info 13)))
    (cond
      ((/= info 0)
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DGBMV" info)
       (go end_label)))
    (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
        (go end_label))
    (cond
      ((char-equal trans #\N)
       (setf lenx n)
       (setf leny m)
       (t
        (setf lenx m)
        (setf leny n)))
      (cond
        ((> incx 0)
         (setf kx 1))
        (t
         (setf kx
          (the fixnum (- 1
                        (f2cl-lib:int-mul
                         (the fixnum (1- lenx))
                         incx))))))
        (cond
          ((> incy 0)
           (setf ky 1))
          (t
           (setf ky
            (the fixnum (- 1
                          (f2cl-lib:int-mul
                           (the fixnum (1- leny))
                           incy))))
          (cond
            ((/= beta one)
             (cond
              ((= incy 1)
               (cond
                ((= beta zero)
                 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                               ((> i leny) nil)
                 (tagbody
                  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                        zero))))
                (t

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i leny) nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (* beta
      (f2cl-lib:fref y-%data%
        (i)
        ((1 *))
        y-%offset%))))))
(t
  (setf iy ky)
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i leny) nil)
      (tagbody
        (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
          zero)
        (setf iy (f2cl-lib:int-add iy incy))))))
  (t
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i leny) nil)
    (tagbody
      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (* beta
          (f2cl-lib:fref y-%data%
            (iy)
            ((1 *))
            y-%offset%)))
      (setf iy (f2cl-lib:int-add iy incy))))))
(if (= alpha zero) (go end_label))
(setf kup1 (f2cl-lib:int-add ku 1))
(cond
  ((char-equal trans #\N)
    (setf jx kx)
    (cond
      ((= incy 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
              (setf temp
                (* alpha
                  (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)))
              (setf k (f2cl-lib:int-sub kup1 j))
              (f2cl-lib:fdo (i

```



```

(max (the fixnum 1)
  (the fixnum
    (f2cl-lib:int-add j
      (f2cl-lib:int-sub
        ku))))
(f2cl-lib:int-add i 1))
(> i
  (min (the fixnum m)
    (the fixnum
      (f2cl-lib:int-add j kl))))
nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data%
        (i)
        ((1 *))
        y-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add k i) j)
          ((1 lda) (1 *))
          a-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)))
        (setf iy ky)
        (setf k (f2cl-lib:int-sub kup1 j))
        (f2cl-lib:fdo (i
          (max (the fixnum 1)
            (the fixnum
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub
                  ku))))
            (f2cl-lib:int-add i 1))
          (> i
            (min (the fixnum m)
              (the fixnum
                (f2cl-lib:int-add j kl))))
          nil)

```

```

(tagbody
  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data%
        (iy)
        ((1 *))
        y-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add k i) j)
          ((1 lda) (1 *))
          a-%offset%))))
    (setf iy (f2cl-lib:int-add iy incy))))
  (setf jx (f2cl-lib:int-add jx incx))
  (if (> j ku) (setf ky (f2cl-lib:int-add ky incy))))))
(t
  (setf jy ky)
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf temp zero)
          (setf k (f2cl-lib:int-sub kup1 j))
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    ku))))
              (f2cl-lib:int-add i 1))
            ((> i
              (min (the fixnum m)
                (the fixnum
                  (f2cl-lib:int-add j kl))))
              nil)
            (tagbody
              (setf temp
                (+ temp
                  (*
                    (f2cl-lib:fref a-%data%
                      ((f2cl-lib:int-add k i) j)
                      ((1 lda) (1 *))
                      a-%offset%)
                    (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%))))))
              (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)

```

```

        (* alpha temp)))
      (setf jy (f2cl-lib:int-add jy incy))))))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf temp zero)
          (setf ix kx)
          (setf k (f2cl-lib:int-sub kup1 j))
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    ku))))))
            (f2cl-lib:int-add i 1))
            ((> i
              (min (the fixnum m)
                (the fixnum
                  (f2cl-lib:int-add j kl))))))
              nil)
          (tagbody
            (setf temp
              (+ temp
                (*
                  (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add k i) j)
                    ((1 lda) (1 *))
                    a-%offset%)
                  (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%))))))
            (setf ix (f2cl-lib:int-add ix incx))))
          (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
              (* alpha temp)))
          (setf jy (f2cl-lib:int-add jy incy))
          (if (> j ku) (setf kx (f2cl-lib:int-add kx incx)))))))))
  end_label
  (return
    (values nil nil nil nil nil nil nil nil nil nil nil nil))))

```

dgemv BLAS**— dgemv.input —**

```

)set break resume
)sys rm -f dgemv.output
)spool dgemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgemv.help —

```

=====
dgemv examples
=====

=====
Man Page Details
=====

NAME
    DGEMV - perform one of the matrix-vector operations  $y := \alpha A x + \beta y$ , or  $y := \alpha A' x + \beta y$ ,

SYNOPSIS
    SUBROUTINE DGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
                     BETA, Y, INCY )

    DOUBLE          PRECISION ALPHA, BETA

    INTEGER          INCX, INCY, LDA, M, N

    CHARACTER*1      TRANS

    DOUBLE          PRECISION A( LDA, * ), X( * ), Y( * )

PURPOSE
    DGEMV performs one of the matrix-vector operations

    where alpha and beta are scalars, x and y are vectors and A
    is an m by n matrix.

```

PARAMETERS

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $y := \alpha * A * x + \beta * y$.

TRANS = 'T' or 't' $y := \alpha * A' * x + \beta * y$.

TRANS = 'C' or 'c' $y := \alpha * A' * x + \beta * y$.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).

Before entry, the leading m by n part of the array A

must contain the matrix of coefficients. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, m)$. Unchanged on exit.

X - DOUBLE PRECISION array of DIMENSION at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$ when TRANS = 'N' or 'n' and at least $(1 + (m - 1) * \text{abs}(\text{INCX}))$ otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y - DOUBLE PRECISION array of DIMENSION at least
 (1 + (m - 1) * abs(INCY)) when TRANS = 'N' or 'n'
 and at least (1 + (n - 1) * abs(INCY)) otherwise.
 Before entry with BETA non-zero, the incremented
 array Y must contain the vector y. On exit, Y is
 overwritten by the updated vector y.

INCY - INTEGER.
 On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

— dgemv.f —

```

SUBROUTINE DGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA, BETA
INTEGER           INCX, INCY, LDA, M, N
CHARACTER*1       TRANS
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
* .. Parameters ..
DOUBLE PRECISION  ONE           , ZERO
PARAMETER         ( ONE = 1.0D+0, ZERO = 0.0D+0 )
* .. Local Scalars ..
DOUBLE PRECISION  TEMP
INTEGER           I, INFO, IX, IY, J, JX, JY, KX, KY, LENX, LENY
* .. External Functions ..
LOGICAL           LSAME

```

```

      EXTERNAL          LSAME
*    .. External Subroutines ..
      EXTERNAL          XERBLA
*    .. Intrinsic Functions ..
      INTRINSIC          MAX
*
*    .. Executable Statements ..
*
*    Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( TRANS, 'N' ).AND.
$           .NOT.LSAME( TRANS, 'T' ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 1
      ELSE IF( M.LT.0 ) THEN
          INFO = 2
      ELSE IF( N.LT.0 ) THEN
          INFO = 3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 8
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 11
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DGEMV ', INFO )
          RETURN
      END IF
*
*    Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$       ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$       RETURN
*
*    Set LENX and LENY, the lengths of the vectors x and y, and set
*    up the start points in X and Y.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
          LENX = N
          LENY = M
      ELSE
          LENX = M
          LENY = N
      END IF
      IF( INCX.GT.0 ) THEN
          KX = 1
      ELSE

```

```

      KX = 1 - ( LENX - 1 ) * INCX
    END IF
    IF( INCY.GT.0 ) THEN
      KY = 1
    ELSE
      KY = 1 - ( LENY - 1 ) * INCY
    END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through A.
*
*   First form  y := beta*y.
*
    IF( BETA.NE.ONE ) THEN
      IF( INCY.EQ.1 ) THEN
        IF( BETA.EQ.ZERO ) THEN
          DO 10, I = 1, LENY
            Y( I ) = ZERO
10          CONTINUE
        ELSE
          DO 20, I = 1, LENY
            Y( I ) = BETA*Y( I )
20          CONTINUE
          END IF
        ELSE
          IY = KY
          IF( BETA.EQ.ZERO ) THEN
            DO 30, I = 1, LENY
              Y( IY ) = ZERO
              IY      = IY  + INCY
30            CONTINUE
          ELSE
            DO 40, I = 1, LENY
              Y( IY ) = BETA*Y( IY )
              IY      = IY  + INCY
40            CONTINUE
          END IF
        END IF
      END IF
      IF( ALPHA.EQ.ZERO )
        $ RETURN
      IF( LSAME( TRANS, 'N' ) ) THEN
*
*   Form  y := alpha*A*x + y.
*
*
      JX = KX
      IF( INCY.EQ.1 ) THEN
        DO 60, J = 1, N
          IF( X( JX ).NE.ZERO ) THEN
            TEMP = ALPHA*X( JX )

```



```

        DO 50, I = 1, M
            Y( I ) = Y( I ) + TEMP*A( I, J )
50      CONTINUE
        END IF
        JX = JX + INCX
60      CONTINUE
    ELSE
        DO 80, J = 1, N
            IF( X( JX ).NE.ZERO )THEN
                TEMP = ALPHA*X( JX )
                IY = KY
                DO 70, I = 1, M
                    Y( IY ) = Y( IY ) + TEMP*A( I, J )
                    IY = IY + INCY
70              CONTINUE
                END IF
                JX = JX + INCX
80          CONTINUE
            END IF
        ELSE
*
*          Form y := alpha*A'*x + y.
*
            JY = KY
            IF( INCX.EQ.1 )THEN
                DO 100, J = 1, N
                    TEMP = ZERO
                    DO 90, I = 1, M
                        TEMP = TEMP + A( I, J )*X( I )
90              CONTINUE
                    Y( JY ) = Y( JY ) + ALPHA*TEMP
                    JY = JY + INCY
100             CONTINUE
            ELSE
                DO 120, J = 1, N
                    TEMP = ZERO
                    IX = KX
                    DO 110, I = 1, M
                        TEMP = TEMP + A( I, J )*X( IX )
                        IX = IX + INCX
110             CONTINUE
                    Y( JY ) = Y( JY ) + ALPHA*TEMP
                    JY = JY + INCY
120             CONTINUE
                END IF
            END IF
*
*          RETURN
*
*          End of DGEMV .

```

```
*
  END
```

— BLAS 2 dgemv —

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dgemv (trans m n alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
              (type (double-float) beta alpha)
              (type fixnum incy incx lda n m)
              (type character trans))
    (f2cl-lib:with-multi-array-data
      ((trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
              (lenx 0) (leny 0) (temp 0.0))
        (declare (type fixnum i info ix iy j jx jy kx ky lenx
                          leny)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
          (setf info 1))
         ((< m 0)
          (setf info 2))
         ((< n 0)
          (setf info 3))
         ((< lda (max (the fixnum 1) (the fixnum m)))
          (setf info 6))
         ((= incx 0)
          (setf info 8))
         ((= incy 0)
          (setf info 11)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DGEMV" info)
          (go end_label)))
        (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
```

```

        (go end_label))
(cond
  ((char-equal trans #\N)
   (setf lenx n)
   (setf leny m)
   (t
    (setf lenx m)
    (setf leny n)))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
    (f2cl-lib:int-sub 1
     (f2cl-lib:int-mul
      (f2cl-lib:int-sub lenx 1)
      incx))))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
    (f2cl-lib:int-sub 1
     (f2cl-lib:int-mul
      (f2cl-lib:int-sub leny 1)
      incy))))))
(cond
  ((/= beta one)
   (cond
    ((= incy 1)
     (cond
      ((= beta zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i leny) nil)
        (tagbody
         (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
          zero))))
      (t
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i leny) nil)
        (tagbody
         (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
          (* beta
           (f2cl-lib:fref y-%data%
            (i)
            ((1 *))
            y-%offset%))))))))))
  (t
   (setf iy ky)
   (cond

```

```

(= beta zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i leny) nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      zero)
    (setf iy (f2cl-lib:int-add iy incy))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i leny) nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (* beta
        (f2cl-lib:fref y-%data%
          (iy)
          ((1 *))
          y-%offset%)))
    (setf iy (f2cl-lib:int-add iy incy))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal trans #\N)
    (setf jx kx)
    (cond
      ((= incy 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
              (setf temp
                (* alpha
                  (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)))
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                    (+
                      (f2cl-lib:fref y-%data%
                        (i)
                        ((1 *))
                        y-%offset%)
                      (* temp
                        (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
                  (setf jx (f2cl-lib:int-add jx incx))))))

```

```
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)))
      (setf iy ky)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
          (+
            (f2cl-lib:fref y-%data%
              (iy)
              ((1 *))
              y-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))))
        (setf iy (f2cl-lib:int-add iy incy))))
      (setf jx (f2cl-lib:int-add jx incx))))))
(t
(setf jy ky)
(cond
  ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (setf temp zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
              (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
```

```

                                x-%offset%))))))
    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
          (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
              (* alpha temp)))
    (setf jy (f2cl-lib:int-add jy incy))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               (> j n) nil)
 (tagbody
  (setf temp zero)
  (setf ix kx)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i m) nil)
    (tagbody
     (setf temp
              (+ temp
                  (*
                   (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                   (f2cl-lib:fref x-%data%
                                   (ix)
                                   ((1 *))
                                   x-%offset%))))))
     (setf ix (f2cl-lib:int-add ix incx)))
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
            (* alpha temp)))
  (setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))

```

dger BLAS

— dger.input —

```

)set break resume
)sys rm -f dger.output
)spool dger.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— dger.help —

```
=====
dger examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGER - perform the rank 1 operation $A := \alpha x y' + A$,

SYNOPSIS

SUBROUTINE DGER (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)

DOUBLE PRECISION ALPHA

INTEGER INCX, INCY, LDA, M, N

DOUBLE PRECISION A(LDA, *), X(*), Y(*)

PURPOSE

DGER performs the rank 1 operation

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

PARAMETERS

M - INTEGER.

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least

(1 + (m - 1) * abs(INCX)). Before entry, the incremented array X must contain the m element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

Y - DOUBLE PRECISION array of dimension at least (1 + (n - 1) * abs(INCY)). Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.

INCY - INTEGER.

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n). Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

— dger.f —

```

SUBROUTINE DGER ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
*   .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, INCY, LDA, M, N
*   .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*      Jack Dongarra, Argonne National Lab.
*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.

```



```

*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
      DOUBLE PRECISION    ZERO
      PARAMETER            ( ZERO = 0.0D+0 )
*      .. Local Scalars ..
      DOUBLE PRECISION    TEMP
      INTEGER              I, INFO, IX, J, JY, KX
*      .. External Subroutines ..
      EXTERNAL            XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC            MAX
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( M.LT.0 )THEN
          INFO = 1
      ELSE IF( N.LT.0 )THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 5
      ELSE IF( INCY.EQ.0 )THEN
          INFO = 7
      ELSE IF( LDA.LT.MAX( 1, M ) )THEN
          INFO = 9
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'DGER ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through A.
*
      IF( INCY.GT.0 )THEN
          JY = 1
      ELSE
          JY = 1 - ( N - 1 )*INCY
      END IF
      IF( INCX.EQ.1 )THEN
          DO 20, J = 1, N

```

```

        IF( Y( JY ).NE.ZERO )THEN
            TEMP = ALPHA*Y( JY )
            DO 10, I = 1, M
                A( I, J ) = A( I, J ) + X( I )*TEMP
10          CONTINUE
            END IF
            JY = JY + INCY
20        CONTINUE
        ELSE
            IF( INCX.GT.0 )THEN
                KX = 1
            ELSE
                KX = 1 - ( M - 1 )*INCX
            END IF
            DO 40, J = 1, N
                IF( Y( JY ).NE.ZERO )THEN
                    TEMP = ALPHA*Y( JY )
                    IX = KX
                    DO 30, I = 1, M
                        A( I, J ) = A( I, J ) + X( IX )*TEMP
                        IX = IX + INCX
30          CONTINUE
                    END IF
                    JY = JY + INCY
40          CONTINUE
            END IF
*
            RETURN
*
*      End of DGER .
*
            END

```

— BLAS 2 dger —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dger (m n alpha x incx y incy a lda)
    (declare (type (simple-array double-float (*)) a y x)
              (type (double-float) alpha)
              (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
      ((x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%)
       (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp 0.0))

```

```

(declare (type fixnum i info ix j jy kx)
         (type (double-float) temp))
(setf info 0)
(cond
  ((< m 0)
   (setf info 1))
  ((< n 0)
   (setf info 2))
  ((= incx 0)
   (setf info 5))
  ((= incy 0)
   (setf info 7))
  ((< lda (max (the fixnum 1) (the fixnum m)))
   (setf info 9)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGER" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
(cond
  ((> incy 0)
   (setf jy 1))
  (t
   (setf jy
    (f2cl-lib:int-sub 1
                      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                          incy)))))
(cond
  ((= incx 1)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 ((> j n) nil)
   (tagbody
    (cond
      ((/= (f2cl-lib:fref y (jy) ((1 *)) zero)
       (setf temp
        (* alpha
         (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)
       (tagbody
        (setf (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *))
                              a-%offset%)
              (+
               (f2cl-lib:fref a-%data%
                               (i j)
                               ((1 lda) (1 *))

```

```

                                a-%offset%)
                                (*
                                (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                                temp))))))
                                (setf jy (f2cl-lib:int-add jy incy))))))
(t
(cond
  (> incx 0)
  (setf kx 1))
(t
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub m 1)
        incx))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref y (jy) ((1 *)) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
      (setf ix kx)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
          (+
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
              temp)))
          (setf ix (f2cl-lib:int-add ix incx))))))
      (setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

dsbmv BLAS**— dsbmv.input —**

```

)set break resume
)sys rm -f dsbmv.output
)spool dsbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dsbmv.help —

```

=====
dsbmv examples
=====

=====
Man Page Details
=====

NAME
    DSBMV - perform the matrix-vector operation  $y := \alpha * A * x + \beta * y$ ,

SYNOPSIS
    SUBROUTINE DSBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA,
                      Y, INCY )

    DOUBLE          PRECISION ALPHA, BETA

    INTEGER          INCX, INCY, K, LDA, N

    CHARACTER*1      UPLO

    DOUBLE          PRECISION A( LDA, * ), X( * ), Y( * )

PURPOSE
    DSBMV performs the matrix-vector operation

    where alpha and beta are scalars, x and y are n element vec-
    tors and A is an n by n symmetric band matrix, with k
    super-diagonals.

```

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy $0 \leq K$. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).

Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N
  M = K + 1 - J
  DO 10, I = MAX( 1, J - K ), J
    A( M + I, J ) = matrix( I, J )
  10 CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of

the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
) A( M + I, J ) = matrix( I, J ) 10    CONTINUE 20
CONTINUE
```

Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $(k + 1)$. Unchanged on exit.

X - DOUBLE PRECISION array of DIMENSION at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the vector x . Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X . INCX must not be zero. Unchanged on exit.

BETA - DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. Unchanged on exit.

Y - DOUBLE PRECISION array of DIMENSION at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array Y must contain the vector y . On exit, Y is overwritten by the updated vector y .

INCY - INTEGER.

On entry, INCY specifies the increment for the elements of Y . INCY must not be zero. Unchanged on exit.

— dsbm.f —

SUBROUTINE DSBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX,

```

$          BETA, Y, INCY )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA, BETA
INTEGER           INCX, INCY, K, LDA, N
CHARACTER*1       UPLO
*
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
DOUBLE PRECISION  ONE          , ZERO
PARAMETER         ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
* .. Local Scalars ..
DOUBLE PRECISION  TEMP1, TEMP2
INTEGER           I, INFO, IX, IY, J, JX, JY, KPLUS1, KX, KY, L
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC         MAX, MIN
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
*
INFO = 0
IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$        .NOT.LSAME( UPLO, 'L' ) ) THEN
    INFO = 1
ELSE IF( N.LT.0 ) THEN
    INFO = 2
ELSE IF( K.LT.0 ) THEN
    INFO = 3
ELSE IF( LDA.LT.( K + 1 ) ) THEN
    INFO = 6
ELSE IF( INCX.EQ.0 ) THEN
    INFO = 8
ELSE IF( INCY.EQ.0 ) THEN
    INFO = 11

```



```

      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'DSBMV ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   Set up the start points in X and Y.
*
      IF( INCX.GT.0 )THEN
        KX = 1
      ELSE
        KX = 1 - ( N - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( N - 1 )*INCY
      END IF
*
*   Start the operations. In this version the elements of the array A
*   are accessed sequentially with one pass through A.
*
*   First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, N
              Y( I ) = BETA*Y( I )
20          CONTINUE
          END IF
        ELSE
          IY = KY
          IF( BETA.EQ.ZERO )THEN
            DO 30, I = 1, N
              Y( IY ) = ZERO
              IY = IY + INCY
30          CONTINUE
          ELSE
            DO 40, I = 1, N
              Y( IY ) = BETA*Y( IY )

```

```

        IY      = IY      + INCY
40      CONTINUE
      END IF
    END IF
  END IF
  IF( ALPHA.EQ.ZERO )
$    RETURN
    IF( LSAME( UPLO, 'U' ) )THEN
*
*      Form y when upper triangle of A is stored.
*
      KPLUS1 = K + 1
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          L      = KPLUS1 - J
          DO 50, I = MAX( 1, J - K ), J - 1
            Y( I ) = Y( I ) + TEMP1*A( L + I, J )
            TEMP2 = TEMP2 + A( L + I, J )*X( I )
50      CONTINUE
          Y( J ) = Y( J ) + TEMP1*A( KPLUS1, J ) + ALPHA*TEMP2
60      CONTINUE
        ELSE
          JX = KX
          JY = KY
          DO 80, J = 1, N
            TEMP1 = ALPHA*X( JX )
            TEMP2 = ZERO
            IX = KX
            IY = KY
            L = KPLUS1 - J
            DO 70, I = MAX( 1, J - K ), J - 1
              Y( IY ) = Y( IY ) + TEMP1*A( L + I, J )
              TEMP2 = TEMP2 + A( L + I, J )*X( IX )
              IX = IX + INCX
              IY = IY + INCY
70      CONTINUE
              Y( JY ) = Y( JY ) + TEMP1*A( KPLUS1, J ) + ALPHA*TEMP2
              JX = JX + INCX
              JY = JY + INCY
              IF( J.GT.K )THEN
                KX = KX + INCX
                KY = KY + INCY
              END IF
80      CONTINUE
          END IF
        ELSE
*
*      Form y when lower triangle of A is stored.

```

```

*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 100, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          Y( J ) = Y( J )      + TEMP1*A( 1, J )
          L      = 1          - J
          DO 90, I = J + 1, MIN( N, J + K )
            Y( I ) = Y( I ) + TEMP1*A( L + I, J )
            TEMP2 = TEMP2 + A( L + I, J )*X( I )
          90    CONTINUE
          Y( J ) = Y( J ) + ALPHA*TEMP2
        100    CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 120, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          Y( JY ) = Y( JY )      + TEMP1*A( 1, J )
          L      = 1          - J
          IX     = JX
          IY     = JY
          DO 110, I = J + 1, MIN( N, J + K )
            IX = IX + INCX
            IY = IY + INCY
            Y( IY ) = Y( IY ) + TEMP1*A( L + I, J )
            TEMP2 = TEMP2 + A( L + I, J )*X( IX )
          110    CONTINUE
            Y( JY ) = Y( JY ) + ALPHA*TEMP2
            JX = JX + INCX
            JY = JY + INCY
          120    CONTINUE
        END IF
      END IF
*
      RETURN
*
*   End of DSBMV .
*
      END

```

— BLAS 2 dsbmv —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)

```

```

        (type (double-float 0.0 0.0) zero))
(defun dsbmv (uplo n k alpha a lda x incx beta y incy)
  (declare (type (simple-array double-float (*)) y x a)
    (type (double-float) beta alpha)
    (type fixnum incy incx lda k n)
    (type character uplo))
  (f2cl-lib:with-multi-array-data
    ((uplo character uplo-%data% uplo-%offset%)
     (a double-float a-%data% a-%offset%)
     (x double-float x-%data% x-%offset%)
     (y double-float y-%data% y-%offset%))
    (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kplus1 0) (kx 0)
      (ky 0) (l 0) (temp1 0.0) (temp2 0.0))
      (declare (type fixnum i info ix iy j jx jy kplus1 kx ky l)
        (type (double-float) temp1 temp2))
      (setf info 0)
      (cond
        ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
         (setf info 1))
        ((< n 0)
         (setf info 2))
        ((< k 0)
         (setf info 3))
        ((< lda (f2cl-lib:int-add k 1))
         (setf info 6))
        ((= incx 0)
         (setf info 8))
        ((= incy 0)
         (setf info 11)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DSBMV" info)
         (go end_label)))
      (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
      (cond
        ((> incx 0)
         (setf kx 1))
        (t
         (setf kx
          (f2cl-lib:int-sub 1
            (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
              incx))))))
      (cond
        ((> incy 0)
         (setf ky 1))
        (t
         (setf ky
          (f2cl-lib:int-sub 1
            (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
              incx))))))

```

```

(f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
  incy))))))
(cond
  ((/= beta one)
    (cond
      ((= incy 1)
        (cond
          ((= beta zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
                zero))))
          (t
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                (* beta
                  (f2cl-lib:fref y-%data%
                    (i)
                    ((1 *))
                    y-%offset%))))))))
      (t
        (setf iy ky)
        (cond
          ((= beta zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
                zero)
              (setf iy (f2cl-lib:int-add iy incy))))))
          (t
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                (* beta
                  (f2cl-lib:fref y-%data%
                    (iy)
                    ((1 *))
                    y-%offset%)))
              (setf iy (f2cl-lib:int-add iy incy))))))))))
    (if (= alpha zero) (go end_label))
  (cond
    ((char-equal uplo #\U)
      (setf kplus1 (f2cl-lib:int-add k 1))
      (cond
        ((and (= incx 1) (= incy 1))

```

```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf temp1
    (* alpha
      (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
  (setf temp2 zero)
  (setf l (f2cl-lib:int-sub kplus1 j))
  (f2cl-lib:fdo (i
    (max (the fixnum 1)
      (the fixnum
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub
            k))))
    (f2cl-lib:int-add i 1))
    (> i
      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    nil)
    (tagbody
      (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
        (+
          (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
          (* temp1
            (f2cl-lib:fref a-%data%
              ((f2cl-lib:int-add 1 i) j)
              ((1 lda) (1 *))
              a-%offset%))))
        (setf temp2
          (+ temp2
            (*
              (f2cl-lib:fref a-%data%
                ((f2cl-lib:int-add 1 i) j)
                ((1 lda) (1 *))
                a-%offset%)
              (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
                x-%offset%))))))
        (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
            (* temp1
              (f2cl-lib:fref a-%data%
                (kplus1 j)
                ((1 lda) (1 *))
                a-%offset%))
            (* alpha temp2))))))
  (t
    (setf jx kx)
    (setf jy ky)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

        (< j n) nil)
(tagbody
  (setf temp1
    (* alpha
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
  (setf temp2 zero)
  (setf ix kx)
  (setf iy ky)
  (setf l (f2cl-lib:int-sub kplus1 j))
  (f2cl-lib:fdo (i
    (max (the fixnum 1)
      (the fixnum
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub
            k))))))
    (f2cl-lib:int-add i 1))
    (< i
      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (+
        (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add 1 i) j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add 1 i) j)
            ((1 lda) (1 *))
            a-%offset%)
          (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%))))))
    (setf ix (f2cl-lib:int-add ix incx))
    (setf iy (f2cl-lib:int-add iy incy)))
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          (kplus1 j)
          ((1 lda) (1 *))
          a-%offset%))
        (* alpha temp2)))
  (setf jx (f2cl-lib:int-add jx incx))

```

```

(setf jy (f2cl-lib:int-add jy incy))
(cond
  (> j k)
    (setf kx (f2cl-lib:int-add kx incx))
    (setf ky (f2cl-lib:int-add ky incy)))))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp1
          (* alpha
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
        (setf temp2 zero)
        (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
            (* temp1
              (f2cl-lib:fref a-%data%
                (1 j)
                ((1 lda) (1 *))
                a-%offset%))))))
        (setf l (f2cl-lib:int-sub 1 j))
        (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
          (f2cl-lib:int-add i 1))
          (> i
            (min (the fixnum n)
              (the fixnum
                (f2cl-lib:int-add j k))))
          nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
            (+
              (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
              (* temp1
                (f2cl-lib:fref a-%data%
                  ((f2cl-lib:int-add 1 i) j)
                  ((1 lda) (1 *))
                  a-%offset%))))))
          (setf temp2
            (+ temp2
              (*
                (f2cl-lib:fref a-%data%
                  ((f2cl-lib:int-add 1 i) j)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%)))))))))

```



```
(t
(setf jx kx)
(setf jy ky)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)

(tagbody
  (setf temp1
    (* alpha
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
  (setf temp2 zero)
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          (1 j)
          ((1 lda) (1 *))
          a-%offset%))))
  (setf l (f2cl-lib:int-sub 1 j))
  (setf ix jx)
  (setf iy jy)
  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
    (f2cl-lib:int-add i 1))
    (> i
      (min (the fixnum n)
        (the fixnum
          (f2cl-lib:int-add j k))))
    nil)

  (tagbody
    (setf ix (f2cl-lib:int-add ix incx))
    (setf iy (f2cl-lib:int-add iy incy))
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (+
        (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add l i) j)
            ((1 lda) (1 *))
            a-%offset%))))

    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add l i) j)
            ((1 lda) (1 *))
            a-%offset%)
          (f2cl-lib:fref x-%data%
            (ix)

```

```

((1 *))
x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
          (* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))

```

dspmv BLAS

— dspmv.input —

```

)set break resume
)sys rm -f dspmv.output
)spool dspmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dspmv.help —

```

=====
dspmv examples
=====

=====
Man Page Details
=====

NAME
    DSPMV - perform the matrix-vector operation    y := alpha*A*x
    + beta*y,

SYNOPSIS
    SUBROUTINE DSPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y,
                      INCY )

```

```

DOUBLE      PRECISION ALPHA, BETA

INTEGER     INCX, INCY, N

CHARACTER*1  UPLO

DOUBLE      PRECISION AP( * ), X( * ), Y( * )

```

PURPOSE

DSPMV performs the matrix-vector operation

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

AP - DOUBLE PRECISION array of DIMENSION at least
 ((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2)

respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(

3, 1) respectively, and so on. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least
 (1 + (n - 1) * abs(INCX)). Before entry, the
 incremented array X must contain the n element vector
 x. Unchanged on exit.

INCX - INTEGER.
 On entry, INCX specifies the increment for the ele-
 ments of X. INCX must not be zero. Unchanged on
 exit.

BETA - DOUBLE PRECISION.
 On entry, BETA specifies the scalar beta. When BETA
 is supplied as zero then Y need not be set on input.
 Unchanged on exit.

Y - DOUBLE PRECISION array of dimension at least
 (1 + (n - 1) * abs(INCY)). Before entry, the
 incremented array Y must contain the n element vector
 y. On exit, Y is overwritten by the updated vector y.

INCY - INTEGER.
 On entry, INCY specifies the increment for the ele-
 ments of Y. INCY must not be zero. Unchanged on
 exit.

— dspmvm.f —

```

SUBROUTINE DSPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
*   .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA, BETA
INTEGER           INCX, INCY, N
CHARACTER*1       UPLO
*   .. Array Arguments ..
DOUBLE PRECISION  AP( * ), X( * ), Y( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.

```

```

*
*
*   .. Parameters ..
DOUBLE PRECISION    ONE          , ZERO
PARAMETER           ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*   .. Local Scalars ..
DOUBLE PRECISION    TEMP1, TEMP2
INTEGER             I, INFO, IX, IY, J, JX, JY, K, KK, KX, KY
*   .. External Functions ..
LOGICAL             LSAME
EXTERNAL            LSAME
*   .. External Subroutines ..
EXTERNAL            XERBLA
*
*   .. Executable Statements ..
*
*   Test the input parameters.
*
    INFO = 0
    IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$         .NOT.LSAME( UPLO, 'L' ) ) THEN
        INFO = 1
    ELSE IF( N.LT.0 ) THEN
        INFO = 2
    ELSE IF( INCX.EQ.0 ) THEN
        INFO = 6
    ELSE IF( INCY.EQ.0 ) THEN
        INFO = 9
    END IF
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DSPMV ', INFO )
        RETURN
    END IF
*
*   Quick return if possible.
*
    IF( ( N.EQ.0 ) .OR. ( ( ALPHA.EQ.ZERO ) .AND. ( BETA.EQ.ONE ) ) )
$    RETURN
*
*   Set up the start points in X and Y.
*
    IF( INCX.GT.0 ) THEN
        KX = 1
    ELSE
        KX = 1 - ( N - 1 ) * INCX
    END IF
    IF( INCY.GT.0 ) THEN
        KY = 1
    ELSE
        KY = 1 - ( N - 1 ) * INCY

```

```

      END IF
*
*   Start the operations. In this version the elements of the array AP
*   are accessed sequentially with one pass through AP.
*
*   First form  $y := \beta y$ .
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, N
              Y( I ) = BETA*Y( I )
20          CONTINUE
          END IF
        ELSE
          IY = KY
          IF( BETA.EQ.ZERO )THEN
            DO 30, I = 1, N
              Y( IY ) = ZERO
              IY      = IY  + INCY
30          CONTINUE
          ELSE
            DO 40, I = 1, N
              Y( IY ) = BETA*Y( IY )
              IY      = IY  + INCY
40          CONTINUE
          END IF
        END IF
      END IF
      IF( ALPHA.EQ.ZERO )
        $ RETURN
      KK = 1
      IF( LSAME( UPLO, 'U' ) )THEN
*
*   Form  $y$  when AP contains the upper triangle.
*
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          K      = KK
          DO 50, I = 1, J - 1
            Y( I ) = Y( I ) + TEMP1*AP( K )
            TEMP2  = TEMP2  + AP( K )*X( I )
            K      = K      + 1
50          CONTINUE

```

```

        Y( J ) = Y( J ) + TEMP1*AP( KK + J - 1 ) + ALPHA*TEMP2
        KK      = KK      + J
60      CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          IX    = KX
          IY    = KY
          DO 70, K = KK, KK + J - 2
            Y( IY ) = Y( IY ) + TEMP1*AP( K )
            TEMP2    = TEMP2  + AP( K )*X( IX )
            IX       = IX     + INCX
            IY       = IY     + INCY
          70      CONTINUE
          Y( JY ) = Y( JY ) + TEMP1*AP( KK + J - 1 ) + ALPHA*TEMP2
          JX      = JX      + INCX
          JY      = JY      + INCY
          KK      = KK      + J
        80      CONTINUE
      END IF
    ELSE
*
*      Form y when AP contains the lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 100, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          Y( J ) = Y( J )      + TEMP1*AP( KK )
          K      = KK      + 1
          DO 90, I = J + 1, N
            Y( I ) = Y( I ) + TEMP1*AP( K )
            TEMP2  = TEMP2  + AP( K )*X( I )
            K      = K      + 1
          90      CONTINUE
          Y( J ) = Y( J ) + ALPHA*TEMP2
          KK     = KK     + ( N - J + 1 )
        100     CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 120, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          Y( JY ) = Y( JY )      + TEMP1*AP( KK )
          IX      = JX
          IY      = JY

```

```

DO 110, K = KK + 1, KK + N - J
    IX      = IX      + INCX
    IY      = IY      + INCY
    Y( IY ) = Y( IY ) + TEMP1*AP( K )
    TEMP2   = TEMP2   + AP( K )*X( IX )
110    CONTINUE
    Y( JY ) = Y( JY ) + ALPHA*TEMP2
    JX      = JX      + INCX
    JY      = JY      + INCY
    KK      = KK      + ( N - J + 1 )
120    CONTINUE
END IF
END IF
*
RETURN
*
*   End of DSPMV .
*
END

```

— BLAS 2 dspmv —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dspmv (uplo n alpha ap x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x ap)
              (type (double-float) beta alpha)
              (type fixnum incy incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (ap double-float ap-%data% ap-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
              (kx 0) (ky 0) (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                  (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          (< n 0)
            (setf info 2))
          (= incx 0)

```



```

    (setf info 6))
    ((= incy 0)
     (setf info 9)))
  (cond
   ((/= info 0)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DSPMV" info)
    (go end_label)))
  (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
  (cond
   ((> incx 0)
    (setf kx 1))
   (t
    (setf kx
      (f2cl-lib:int-sub 1
        (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
          incx))))))
  (cond
   ((> incy 0)
    (setf ky 1))
   (t
    (setf ky
      (f2cl-lib:int-sub 1
        (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
          incy))))))
  (cond
   ((/= beta one)
    (cond
     ((= incy 1)
      (cond
       ((= beta zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i n) nil)
          (tagbody
           (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
             zero))))))
       (t
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i n) nil)
          (tagbody
           (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
             (* beta
              (f2cl-lib:fref y-%data%
                (i)
                ((1 *))
                y-%offset%))))))))))
   (t
    (setf iy ky)
    (cond

```

```

(= beta zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)

(tagbody
  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
    zero)
  (setf iy (f2cl-lib:int-add iy incy))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (* beta
        (f2cl-lib:fref y-%data%
          (iy)
          ((1 *))
          y-%offset%)))
    (setf iy (f2cl-lib:int-add iy incy))))))
(if (= alpha zero) (go end_label))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
    (cond
      ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (setf temp1
            (* alpha
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf k kk)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i
              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
            nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
              (+
                (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                (* temp1
                  (f2cl-lib:fref ap-%data%
                    (k)
                    ((1 *))
                    ap-%offset%))))
            (setf temp2
              (+ temp2
                (*
                  (f2cl-lib:fref ap-%data%
                    (k)

```

```

                                ((1 *))
                                ap-%offset%)
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%))))
(setf k (f2cl-lib:int-add k 1)))
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
(+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
(* temp1
(f2cl-lib:fref ap-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add kk j)
1))
((1 *))
ap-%offset%))
(* alpha temp2)))
(setf kk (f2cl-lib:int-add kk j))))))
(t
(setf jx kx)
(setf jy ky)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(setf temp1
(* alpha
(f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
(setf temp2 zero)
(setf ix kx)
(setf iy ky)
(f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
(> k
(f2cl-lib:int-add kk
j
(f2cl-lib:int-sub 2)))
nil)
(tagbody
(setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
(+
(f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
(* temp1
(f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%))))
(setf temp2
(+ temp2
(*
(f2cl-lib:fref ap-%data%
(k)

```

```

((1 *))
ap-%offset%)
(f2cl-lib:fref x-%data%
(ix)
((1 *))
x-%offset%))))
(setf ix (f2cl-lib:int-add ix incx))
(setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
(+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
(* temp1
(f2cl-lib:fref ap-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add kk j)
1))
((1 *))
ap-%offset%))
(* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(setf kk (f2cl-lib:int-add kk j))))))
(t
(cond
((and (= incx 1) (= incy 1))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(setf temp1
(* alpha
(f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
(setf temp2 zero)
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
(+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
(* temp1
(f2cl-lib:fref ap-%data%
(kk)
((1 *))
ap-%offset%))))
(setf k (f2cl-lib:int-add kk 1))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
(f2cl-lib:int-add i 1))
(> i n) nil)
(tagbody
(setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
(+
(f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
(* temp1
(f2cl-lib:fref ap-%data%
(k)
((1 *))

```

```

                                ap-%offset%))))
  (setf temp2
    (+ temp2
      (*
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))
  (setf k (f2cl-lib:int-add k 1)))
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (* alpha temp2)))
(setf kk
  (f2cl-lib:int-add kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
      (setf temp2 zero)
      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
          (* temp1
            (f2cl-lib:fref ap-%data%
              (kk)
              ((1 *))
              ap-%offset%))))))
      (setf ix jx)
      (setf iy jy)
      (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
        (f2cl-lib:int-add k 1))
        ((> k
          (f2cl-lib:int-add kk
            n
            (f2cl-lib:int-sub j))))
        nil)
      (tagbody
        (setf ix (f2cl-lib:int-add ix incx))
        (setf iy (f2cl-lib:int-add iy incy))

```

```

      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (+
          (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
          (* temp1
            (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%))))))
    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
          (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%))))))
    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (* alpha temp2)))
    (setf jx (f2cl-lib:int-add jx incx))
    (setf jy (f2cl-lib:int-add jy incy))
    (setf kk
      (f2cl-lib:int-add kk
        (f2cl-lib:int-add
          (f2cl-lib:int-sub n j)
          1))))))
  end_label
  (return (values nil nil nil nil nil nil nil nil))))

```

dspr2 BLAS

— dspr2.input —

```

)set break resume
)sys rm -f dspr2.output
)spool dspr2.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— dspr2.help —

=====

dspr2 examples

=====

=====

Man Page Details

=====

NAME

DSPR2 - perform the symmetric rank 2 operation $A := \alpha x x' + \alpha y y' + A$,

SYNOPSIS

SUBROUTINE DSPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)

DOUBLE PRECISION ALPHA

INTEGER INCX, INCY, N

CHARACTER*1 UPLO

DOUBLE PRECISION AP(*), X(*), Y(*)

PURPOSE

DSPR2 performs the symmetric rank 2 operation

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

- N** - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.
- ALPHA** - DOUBLE PRECISION.
On entry, ALPHA specifies the scalar alpha. Unchanged on exit.
- X** - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.
- INCX** - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.
- Y** - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCY}))$. Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.
- INCY** - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.
- AP** - DOUBLE PRECISION array of DIMENSION at least $((n * (n + 1)) / 2)$. Before entry with `UPLO = 'U'` or `'u'`, the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that `AP(1)` contains `a(1, 1)`, `AP(2)` and `AP(3)` contain `a(1, 2)` and `a(2, 2)` respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with `UPLO = 'L'` or `'l'`, the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that `AP(1)` contains `a(1, 1)`, `AP(2)` and `AP(3)` contain `a(2, 1)` and `a(3, 1)` respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.
-

— dspr2.f —

```

SUBROUTINE DSPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, INCY, N
CHARACTER*1       UPLO
*
* .. Array Arguments ..
DOUBLE PRECISION  AP( * ), X( * ), Y( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
DOUBLE PRECISION  ZERO
PARAMETER         ( ZERO = 0.0D+0 )
*
* .. Local Scalars ..
DOUBLE PRECISION  TEMP1, TEMP2
INTEGER           I, INFO, IX, IY, J, JX, JY, K, KK, KX, KY
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
*
INFO = 0
IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$      .NOT.LSAME( UPLO, 'L' ) ) THEN
    INFO = 1
ELSE IF( N.LT.0 ) THEN
    INFO = 2
ELSE IF( INCX.EQ.0 ) THEN
    INFO = 5
ELSE IF( INCY.EQ.0 ) THEN
    INFO = 7
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DSPR2 ', INFO )
    RETURN

```

```

      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*   Set up the start points in X and Y if the increments are not both
*   unity.
*
      IF( ( INCX.NE.1 ).OR.( INCY.NE.1 ) )THEN
        IF( INCX.GT.0 )THEN
          KX = 1
        ELSE
          KX = 1 - ( N - 1 )*INCX
        END IF
        IF( INCY.GT.0 )THEN
          KY = 1
        ELSE
          KY = 1 - ( N - 1 )*INCY
        END IF
        JX = KX
        JY = KY
      END IF
*
*   Start the operations. In this version the elements of the array AP
*   are accessed sequentially with one pass through AP.
*
      KK = 1
      IF( LSAME( UPLO, 'U' ) )THEN
*
*       Form A when upper triangle is stored in AP.
*
        IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
          DO 20, J = 1, N
            IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
              TEMP1 = ALPHA*Y( J )
              TEMP2 = ALPHA*X( J )
              K      = KK
              DO 10, I = 1, J
                AP( K ) = AP( K ) + X( I )*TEMP1 + Y( I )*TEMP2
                K      = K      + 1
10              CONTINUE
              END IF
              KK = KK + J
20            CONTINUE
            ELSE
              DO 40, J = 1, N
                IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                  TEMP1 = ALPHA*Y( JY )

```

```

        TEMP2 = ALPHA*X( JX )
        IX    = KX
        IY    = KY
        DO 30, K = KK, KK + J - 1
            AP( K ) = AP( K ) + X( IX )*TEMP1 + Y( IY )*TEMP2
            IX      = IX      + INCX
            IY      = IY      + INCY
30      CONTINUE
        END IF
        JX = JX + INCX
        JY = JY + INCY
        KK = KK + J
40      CONTINUE
    END IF
ELSE
*
*      Form A when lower triangle is stored in AP.
*
    IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
            IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*Y( J )
                TEMP2 = ALPHA*X( J )
                K      = KK
                DO 50, I = J, N
                    AP( K ) = AP( K ) + X( I )*TEMP1 + Y( I )*TEMP2
                    K      = K      + 1
50      CONTINUE
                END IF
                KK = KK + N - J + 1
60      CONTINUE
            ELSE
                DO 80, J = 1, N
                    IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                        TEMP1 = ALPHA*Y( JY )
                        TEMP2 = ALPHA*X( JX )
                        IX    = JX
                        IY    = JY
                        DO 70, K = KK, KK + N - J
                            AP( K ) = AP( K ) + X( IX )*TEMP1 + Y( IY )*TEMP2
                            IX      = IX      + INCX
                            IY      = IY      + INCY
70      CONTINUE
                        END IF
                        JX = JX + INCX
                        JY = JY + INCY
                        KK = KK + N - J + 1
80      CONTINUE
                    END IF
                END IF
            END IF

```

```

*
      RETURN
*
*   End of DSPR2 .
*
      END

```

— BLAS 2 dspr2 —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dspr2 (uplo n alpha x incx y incy ap)
    (declare (type (simple-array double-float (*)) ap y x)
              (type (double-float) alpha)
              (type fixnum incy incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%)
       (ap double-float ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
              (kx 0) (ky 0) (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                  (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((< n 0)
          (setf info 2))
         ((= incx 0)
          (setf info 5))
         ((= incy 0)
          (setf info 7)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DSPR2" info)
          (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
         ((or (/= incx 1) (/= incy 1))
          (cond
           (> incx 0)

```

```

      (setf kx 1))
    (t
      (setf kx
        (f2cl-lib:int-sub 1
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub n 1)
            incx))))))
  (cond
    ((> incy 0)
      (setf ky 1))
    (t
      (setf ky
        (f2cl-lib:int-sub 1
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub n 1)
            incy))))))
  (setf jx kx)
  (setf jy ky))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
    (cond
      ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j n) nil)
          (tagbody
            (cond
              ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (/= (f2cl-lib:fref y (j) ((1 *))) zero))
                (setf temp1
                  (* alpha
                    (f2cl-lib:fref y-%data%
                      (j)
                      ((1 *))
                      y-%offset%)))
                (setf temp2
                  (* alpha
                    (f2cl-lib:fref x-%data%
                      (j)
                      ((1 *))
                      x-%offset%)))
                (setf k kk)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i j) nil)
                  (tagbody
                    (setf (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
                      (+

```

```

(f2cl-lib:fref ap-%data%
  (k)
  ((1 *))
  ap-%offset%)
(*
  (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
  temp1)
(*
  (f2cl-lib:fref y-%data%
    (i)
    ((1 *))
    y-%offset%)
  temp2)))
(setf k (f2cl-lib:int-add k 1))))
(setf kk (f2cl-lib:int-add kk j))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
              (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
          (setf temp1
            (* alpha
              (f2cl-lib:fref y-%data%
                (jy)
                ((1 *))
                y-%offset%)))
          (setf temp2
            (* alpha
              (f2cl-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)))
          (setf ix kx)
          (setf iy ky)
          (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
            ((> k
              (f2cl-lib:int-add kk
                j
                (f2cl-lib:int-sub 1)))
              nil)
            (tagbody
              (setf (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%)

```

```

      (+
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
        (*
          (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%)
          temp1)
        (*
          (f2cl-lib:fref y-%data%
            (iy)
            ((1 *))
            y-%offset%)
          temp2)))
      (setf ix (f2cl-lib:int-add ix incx))
      (setf iy (f2cl-lib:int-add iy incy))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond
          ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
            (/= (f2cl-lib:fref y (j) ((1 *))) zero))
            (setf temp1
              (* alpha
                (f2cl-lib:fref y-%data%
                  (j)
                  ((1 *))
                  y-%offset%)))
            (setf temp2
              (* alpha
                (f2cl-lib:fref x-%data%
                  (j)
                  ((1 *))
                  x-%offset%)))
            (setf k kk)
            (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref ap-%data%
                (k)
                ((1 *)))

```

```

                                ap-%offset%)
(+
  (f2cl-lib:fref ap-%data%
                  (k)
                  ((1 *))
                  ap-%offset%)
  (*
    (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%)
    temp1)
  (*
    (f2cl-lib:fref y-%data%
                    (i)
                    ((1 *))
                    y-%offset%)
    temp2)))
(setf k (f2cl-lib:int-add k 1))))))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                (> j n) nil)
  (tagbody
    (cond
      ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
        (setf temp1
          (* alpha
            (f2cl-lib:fref y-%data%
                            (jy)
                            ((1 *))
                            y-%offset%)))
          (setf temp2
            (* alpha
              (f2cl-lib:fref x-%data%
                              (jx)
                              ((1 *))
                              x-%offset%)))
            (setf ix jx)
            (setf iy jy)
            (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                          (> k
                            (f2cl-lib:int-add kk
                                                  n
                                                  (f2cl-lib:int-sub j))))
              nil)
            nil)

```



```

(tagbody
  (setf (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
        (+
          (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
          (*
            (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
            temp1)
          (*
            (f2cl-lib:fref y-%data%
                          (iy)
                          ((1 *))
                          y-%offset%)
            temp2)))
  (setf ix (f2cl-lib:int-add ix incx))
  (setf iy (f2cl-lib:int-add iy incy))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

dspr BLAS

— dspr.input —

```

)set break resume
)sys rm -f dspr.output
)spool dspr.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— dspr.help —

```
=====
dspr examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DSPR - perform the symmetric rank 1 operation $A := \alpha * x * x' + A$,

SYNOPSIS

SUBROUTINE DSPR (UPLO, N, ALPHA, X, INCX, AP)

DOUBLE PRECISION ALPHA

INTEGER INCX, N

CHARACTER*1 UPLO

DOUBLE PRECISION AP(*), X(*)

PURPOSE

DSPR performs the symmetric rank 1 operation

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.
On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

AP - DOUBLE PRECISION array of DIMENSION at least $((n * (n + 1)) / 2)$. Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix.

— dspr.f —

```

SUBROUTINE DSPR ( UPLO, N, ALPHA, X, INCX, AP )
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, N
CHARACTER*1       UPLO
* .. Array Arguments ..
DOUBLE PRECISION  AP( * ), X( * )

```

```

*      ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
DOUBLE PRECISION    ZERO
PARAMETER            ( ZERO = 0.0D+0 )
* .. Local Scalars ..
DOUBLE PRECISION    TEMP
INTEGER              I, INFO, IX, J, JX, K, KK, KX
* .. External Functions ..
LOGICAL              LSAME
EXTERNAL             LSAME
* .. External Subroutines ..
EXTERNAL             XERBLA
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
    INFO = 0
    IF      ( .NOT.LSAME( UPLO, 'U' ) ).AND.
$          .NOT.LSAME( UPLO, 'L' )          )THEN
        INFO = 1
    ELSE IF( N.LT.0 )THEN
        INFO = 2
    ELSE IF( INCX.EQ.0 )THEN
        INFO = 5
    END IF
    IF( INFO.NE.0 )THEN
        CALL XERBLA( 'DSPR ', INFO )
        RETURN
    END IF
*
* Quick return if possible.
*
    IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
* Set the start point in X if the increment is not unity.
*
    IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 )*INCX

```

```

ELSE IF( INCX.NE.1 )THEN
      KX = 1
END IF

*
* Start the operations. In this version the elements of the array AP
* are accessed sequentially with one pass through AP.
*

KK = 1
IF( LSAME( UPLO, 'U' ) )THEN

*
* Form A when upper triangle is stored in AP.
*

  IF( INCX.EQ.1 )THEN
    DO 20, J = 1, N
      IF( X( J ).NE.ZERO )THEN
        TEMP = ALPHA*X( J )
        K = KK
        DO 10, I = 1, J
          AP( K ) = AP( K ) + X( I )*TEMP
          K = K + 1
10      CONTINUE
        END IF
        KK = KK + J
20    CONTINUE
  ELSE
    JX = KX
    DO 40, J = 1, N
      IF( X( JX ).NE.ZERO )THEN
        TEMP = ALPHA*X( JX )
        IX = KX
        DO 30, K = KK, KK + J - 1
          AP( K ) = AP( K ) + X( IX )*TEMP
          IX = IX + INCX
30      CONTINUE
        END IF
        JX = JX + INCX
        KK = KK + J
40    CONTINUE
  END IF
  ELSE

*
* Form A when lower triangle is stored in AP.
*

  IF( INCX.EQ.1 )THEN
    DO 60, J = 1, N
      IF( X( J ).NE.ZERO )THEN
        TEMP = ALPHA*X( J )
        K = KK
        DO 50, I = J, N
          AP( K ) = AP( K ) + X( I )*TEMP

```

```

          K      = K      + 1
50      CONTINUE
      END IF
      KK = KK + N - J + 1
60      CONTINUE
      ELSE
      JX = KX
      DO 80, J = 1, N
      IF( X( JX ).NE.ZERO )THEN
      TEMP = ALPHA*X( JX )
      IX   = JX
      DO 70, K = KK, KK + N - J
      AP( K ) = AP( K ) + X( IX )*TEMP
      IX      = IX      + INCX
70      CONTINUE
      END IF
      JX = JX + INCX
      KK = KK + N - J + 1
80      CONTINUE
      END IF
      END IF
*
      RETURN
*
*      End of DSPR .
*
      END

```

— BLAS 2 dspr —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dspr (uplo n alpha x incx ap)
    (declare (type (simple-array double-float (*)) ap x)
              (type (double-float) alpha)
              (type fixnum incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x double-float x-%data% x-%offset%)
       (ap double-float ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0) (kx 0) (temp 0.0))
        (declare (type fixnum i info ix j jx k kk kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond

```

```

((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
 (setf info 1))
(< n 0)
(setf info 2))
(= incx 0)
(setf info 5)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DSPR" info)
  (go end_label)))
(if (or (= n 0) (= alpha zero)) (go end_label))
(cond
  (<= incx 0)
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx))))

  (/= incx 1)
  (setf kx 1)))
(setf kk 1)
(cond
  (char-equal uplo #\U)
  (cond
    (= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        (/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref x-%data%
              (j)
              ((1 *))
              x-%offset%)))
        (setf k kk)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i j) nil)
        (tagbody
          (setf (f2cl-lib:fref ap-%data%
            (k)
            ((1 *))
            ap-%offset%)
            (+
              (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%)

```

```

(*
  (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
  temp)))
  (setf k (f2cl-lib:int-add k 1))))))
  (setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)))
        (setf ix kx)
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
          (> k
            (f2cl-lib:int-add kk
              j
              (f2cl-lib:int-sub 1)))
            nil)
          (tagbody
            (setf (f2cl-lib:fref ap-%data%
              (k)
              ((1 *))
              ap-%offset%)
              (+
                (f2cl-lib:fref ap-%data%
                  (k)
                  ((1 *))
                  ap-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%)
                  temp)))
              (setf ix (f2cl-lib:int-add ix incx))))))
            (setf jx (f2cl-lib:int-add jx incx))
            (setf kk (f2cl-lib:int-add kk j))))))
(t
  (cond
    ((= incx 1)

```



```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref x-%data%
                        (j)
                        ((1 *))
                        x-%offset%)))
      (setf k kk)
      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
        (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%)
          (+
            (f2cl-lib:fref ap-%data%
                          (k)
                          ((1 *))
                          ap-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                            (i)
                            ((1 *))
                            x-%offset%)
              temp)))
        (setf k (f2cl-lib:int-add k 1))))))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)))
        (setf ix jx)
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))

```

```

                                (> k
                                (f2cl-lib:int-add kk
                                   n
                                   (f2cl-lib:int-sub j)))
                                nil)
(tagbody
  (setf (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
    (+
      (f2cl-lib:fref ap-%data%
                    (k)
                    ((1 *))
                    ap-%offset%)
      (*
        (f2cl-lib:fref x-%data%
                      (ix)
                      ((1 *))
                      x-%offset%)
        temp)))
  (setf ix (f2cl-lib:int-add ix incx))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))))
end_label
(return (values nil nil nil nil nil nil))))

```

dsymv BLAS

— dsymv.input —

```

)set break resume
)sys rm -f dsymv.output
)spool dsymv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dsymv.help —

=====

dsymv examples

=====

=====

Man Page Details

=====

NAME

DSYMV - perform the matrix-vector operation $y := \alpha * A * x + \beta * y$,

SYNOPSIS

SUBROUTINE DSYMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y,
INCY)

DOUBLE PRECISION ALPHA, BETA

INTEGER INCX, INCY, LDA, N

CHARACTER*1 UPLO

DOUBLE PRECISION A(LDA, *), X(*), Y(*)

PURPOSE

DSYMV performs the matrix-vector operation

where alpha and beta are scalars, x and y are n element vectors and A is an n by n symmetric matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.

- On entry, *N* specifies the order of the matrix *A*. *N* must be at least zero. Unchanged on exit.
- ALPHA** - DOUBLE PRECISION.
On entry, *ALPHA* specifies the scalar *alpha*.
Unchanged on exit.
- A** - DOUBLE PRECISION array of DIMENSION (*LDA*, *n*).
Before entry with *UPLO* = 'U' or 'u', the leading *n* by *n* upper triangular part of the array *A* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *A* is not referenced. Before entry with *UPLO* = 'L' or 'l', the leading *n* by *n* lower triangular part of the array *A* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *A* is not referenced. Unchanged on exit.
- LDA** - INTEGER.
On entry, *LDA* specifies the first dimension of *A* as declared in the calling (sub) program. *LDA* must be at least $\max(1, n)$. Unchanged on exit.
- X** - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array *X* must contain the *n* element vector *x*. Unchanged on exit.
- INCX** - INTEGER.
On entry, *INCX* specifies the increment for the elements of *X*. *INCX* must not be zero. Unchanged on exit.
- BETA** - DOUBLE PRECISION.
On entry, *BETA* specifies the scalar *beta*. When *BETA* is supplied as zero then *Y* need not be set on input. Unchanged on exit.
- Y** - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCY}))$. Before entry, the incremented array *Y* must contain the *n* element vector *y*. On exit, *Y* is overwritten by the updated vector *y*.
- INCY** - INTEGER.
On entry, *INCY* specifies the increment for the elements of *Y*. *INCY* must not be zero. Unchanged on exit.

— dsymv.f —

```

SUBROUTINE DSYMV ( UPLO, N, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA, BETA
INTEGER           INCX, INCY, LDA, N
CHARACTER*1       UPLO
*
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
DOUBLE PRECISION  ONE           , ZERO
PARAMETER         ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
* .. Local Scalars ..
DOUBLE PRECISION  TEMP1, TEMP2
INTEGER           I, INFO, IX, IY, J, JX, JY, KX, KY
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC         MAX
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
*
INFO = 0
IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$        .NOT.LSAME( UPLO, 'L' ) ) THEN
    INFO = 1
ELSE IF( N.LT.0 ) THEN
    INFO = 2
ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
    INFO = 5

```

```

ELSE IF( INCX.EQ.0 )THEN
    INFO = 7
ELSE IF( INCY.EQ.0 )THEN
    INFO = 10
END IF
IF( INFO.NE.0 )THEN
    CALL XERBLA( 'DSYMV ', INFO )
    RETURN
END IF

*
* Quick return if possible.
*
IF( ( N.EQ.0 ).OR.( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$  RETURN

*
* Set up the start points in X and Y.
*
IF( INCX.GT.0 )THEN
    KX = 1
ELSE
    KX = 1 - ( N - 1 )*INCX
END IF
IF( INCY.GT.0 )THEN
    KY = 1
ELSE
    KY = 1 - ( N - 1 )*INCY
END IF

*
* Start the operations. In this version the elements of A are
* accessed sequentially with one pass through the triangular part
* of A.
*
* First form y := beta*y.
*
IF( BETA.NE.ONE )THEN
    IF( INCY.EQ.1 )THEN
        IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
                Y( I ) = ZERO
10          CONTINUE
        ELSE
            DO 20, I = 1, N
                Y( I ) = BETA*Y( I )
20          CONTINUE
        END IF
    ELSE
        IY = KY
        IF( BETA.EQ.ZERO )THEN
            DO 30, I = 1, N
                Y( IY ) = ZERO

```

```

        IY      = IY  + INCY
30      CONTINUE
      ELSE
        DO 40, I = 1, N
          Y( IY ) = BETA*Y( IY )
          IY      = IY      + INCY
40      CONTINUE
      END IF
    END IF
  END IF
  IF( ALPHA.EQ.ZERO )
    $ RETURN
    IF( LSAME( UPLO, 'U' ) )THEN
*
*      Form y when A is stored in upper triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          DO 50, I = 1, J - 1
            Y( I ) = Y( I ) + TEMP1*A( I, J )
            TEMP2 = TEMP2 + A( I, J )*X( I )
50          CONTINUE
          Y( J ) = Y( J ) + TEMP1*A( J, J ) + ALPHA*TEMP2
60        CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          IX = KX
          IY = KY
          DO 70, I = 1, J - 1
            Y( IY ) = Y( IY ) + TEMP1*A( I, J )
            TEMP2 = TEMP2 + A( I, J )*X( IX )
            IX = IX + INCX
            IY = IY + INCY
70          CONTINUE
          Y( JY ) = Y( JY ) + TEMP1*A( J, J ) + ALPHA*TEMP2
          JX = JX + INCX
          JY = JY + INCY
80        CONTINUE
      END IF
    ELSE
*
*      Form y when A is stored in lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN

```

```

DO 100, J = 1, N
  TEMP1 = ALPHA*X( J )
  TEMP2 = ZERO
  Y( J ) = Y( J )      + TEMP1*A( J, J )
  DO 90, I = J + 1, N
    Y( I ) = Y( I ) + TEMP1*A( I, J )
    TEMP2 = TEMP2 + A( I, J )*X( I )
90    CONTINUE
    Y( J ) = Y( J ) + ALPHA*TEMP2
100   CONTINUE
ELSE
  JX = KX
  JY = KY
  DO 120, J = 1, N
    TEMP1 = ALPHA*X( JX )
    TEMP2 = ZERO
    Y( JY ) = Y( JY )      + TEMP1*A( J, J )
    IX = JX
    IY = JY
    DO 110, I = J + 1, N
      IX = IX + INCX
      IY = IY + INCY
      Y( IY ) = Y( IY ) + TEMP1*A( I, J )
      TEMP2 = TEMP2 + A( I, J )*X( IX )
110   CONTINUE
      Y( JY ) = Y( JY ) + ALPHA*TEMP2
      JX = JX + INCX
      JY = JY + INCY
120   CONTINUE
END IF
END IF
*
  RETURN
*
*   End of DSYMV .
*
  END

```

— BLAS 2 dsymv —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dsymv (uplo n alpha a lda x incx beta y incy)
    (declare (type (simple-array double-float (*)) y x a)
              (type (double-float) beta alpha)

```



```

        (type fixnum incy incx lda n)
        (type character uplo))
(f2cl-lib:with-multi-array-data
  ((uplo character uplo-%data% uplo-%offset%)
   (a double-float a-%data% a-%offset%)
   (x double-float x-%data% x-%offset%)
   (y double-float y-%data% y-%offset%))
(prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
      (temp1 0.0) (temp2 0.0))
(declare (type fixnum i info ix iy j jx jy kx ky)
          (type (double-float) temp1 temp2))
(setf info 0)
(cond
  ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
   (setf info 1))
  ((< n 0)
   (setf info 2))
  ((< lda (max (the fixnum 1) (the fixnum n)))
   (setf info 5))
  ((= incx 0)
   (setf info 7))
  ((= incy 0)
   (setf info 10)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DSYMV" info)
   (go end_label)))
(if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
        (f2cl-lib:int-sub 1
                          (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                              incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
        (f2cl-lib:int-sub 1
                          (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                              incy)))))
(cond
  ((/= beta one)
   (cond
    ((= incy 1)

```

```

(cond
  ((= beta zero)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 (> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
                        zero))))
  (t
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 (> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
                        (* beta
                          (f2cl-lib:fref y-%data%
                                           (i)
                                           ((1 *))
                                           y-%offset%)))))))
(t
 (setf iy ky)
 (cond
  ((= beta zero)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 (> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
                        zero)
          (setf iy (f2cl-lib:int-add iy incy))))
  (t
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 (> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
                        (* beta
                          (f2cl-lib:fref y-%data%
                                           (iy)
                                           ((1 *))
                                           y-%offset%)))
          (setf iy (f2cl-lib:int-add iy incy)))))))
(if (= alpha zero) (go end_label))
(cond
 ((char-equal uplo #\U)
  (cond
   ((and (= incx 1) (= incy 1))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  (> j n) nil)
    (tagbody
     (setf temp1
            (* alpha
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
     (setf temp2 zero)

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%))))))
  (setf temp2
    (+ temp2
      (*
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))
  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%))
      (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
    (setf temp2 zero)
    (setf ix kx)
    (setf iy ky)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)

```

```

(+
  (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
  (* temp1
    (f2cl-lib:fref a-%data%
      (i j)
      ((1 lda) (1 *))
      a-%offset%))))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%))))
  (setf ix (f2cl-lib:int-add ix incx))
  (setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* temp1
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%)
      (* alpha temp2)))
  (setf jx (f2cl-lib:int-add jx incx))
  (setf jy (f2cl-lib:int-add jy incy))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf temp1
            (* alpha
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
          (setf temp2 zero)
          (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
            (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
              (* temp1
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
            ((> i n) nil)

```

```

(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%))))))
  (setf temp2
    (+ temp2
      (*
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))
  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
      (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
    (setf temp2 zero)
    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (setf ix jx)
    (setf iy jy)
    (f2cl-lib:fd0 (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf ix (f2cl-lib:int-add ix incx))
      (setf iy (f2cl-lib:int-add iy incy))
      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (+

```

```

(f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
(* temp1
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil))))

```

dsyr2 BLAS

— dsyr2.input —

```

)set break resume
)sys rm -f dsyr2.output
)spool dsyr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dsyr2.help —

=====

dsyr2 examples

=====

Man Page Details

=====

NAME

DSYR2 - perform the symmetric rank 2 operation $A := \alpha x x' + \alpha y y' + A$,

SYNOPSIS

SUBROUTINE DSYR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA
)

DOUBLE PRECISION ALPHA

INTEGER INCX, INCY, LDA, N

CHARACTER*1 UPLO

DOUBLE PRECISION A(LDA, *), X(*), Y(*)

PURPOSE

DSYR2 performs the symmetric rank 2 operation

where alpha is a scalar, x and y are n element vectors and A is an n by n symmetric matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

Y - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCY}))$. Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n). Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

— dsyr2.f —

```
SUBROUTINE DSYR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )
* .. Scalar Arguments ..
```



```

      DOUBLE PRECISION  ALPHA
      INTEGER           INCX, INCY, LDA, N
      CHARACTER*1       UPLO
*    .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), X( * ), Y( * )
*    ..
*
*    Level 2 Blas routine.
*
*    -- Written on 22-October-1986.
*       Jack Dongarra, Argonne National Lab.
*       Jeremy Du Croz, Nag Central Office.
*       Sven Hammarling, Nag Central Office.
*       Richard Hanson, Sandia National Labs.
*
*
*    .. Parameters ..
      DOUBLE PRECISION  ZERO
      PARAMETER         ( ZERO = 0.0D+0 )
*    .. Local Scalars ..
      DOUBLE PRECISION  TEMP1, TEMP2
      INTEGER           I, INFO, IX, IY, J, JX, JY, KX, KY
*    .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL          LSAME
*    .. External Subroutines ..
      EXTERNAL          XERBLA
*    .. Intrinsic Functions ..
      INTRINSIC         MAX
*    ..
*    .. Executable Statements ..
*
*    Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ).AND.
$           .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = 1
      ELSE IF( N.LT.0 ) THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 5
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 7
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = 9
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DSYR2 ', INFO )
          RETURN

```

```

      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*   Set up the start points in X and Y if the increments are not both
*   unity.
*
      IF( ( INCX.NE.1 ).OR.( INCY.NE.1 ) )THEN
        IF( INCX.GT.0 )THEN
          KX = 1
        ELSE
          KX = 1 - ( N - 1 )*INCX
        END IF
        IF( INCY.GT.0 )THEN
          KY = 1
        ELSE
          KY = 1 - ( N - 1 )*INCY
        END IF
        JX = KX
        JY = KY
      END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through the triangular part
*   of A.
*
      IF( LSAME( UPLO, 'U' ) )THEN
*
*       Form A when A is stored in the upper triangle.
*
        IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
          DO 20, J = 1, N
            IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
              TEMP1 = ALPHA*Y( J )
              TEMP2 = ALPHA*X( J )
              DO 10, I = 1, J
                A( I, J ) = A( I, J ) + X( I )*TEMP1 + Y( I )*TEMP2
10              CONTINUE
            END IF
20          CONTINUE
          ELSE
            DO 40, J = 1, N
              IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*Y( JY )
                TEMP2 = ALPHA*X( JX )
                IX = KX
                IY = KY

```

```

        DO 30, I = 1, J
          A( I, J ) = A( I, J ) + X( IX )*TEMP1
$          + Y( IY )*TEMP2
          IX      = IX      + INCX
          IY      = IY      + INCY
30      CONTINUE
        END IF
        JX = JX + INCX
        JY = JY + INCY
40      CONTINUE
      END IF
    ELSE
*
*      Form A when A is stored in the lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
            TEMP1 = ALPHA*Y( J )
            TEMP2 = ALPHA*X( J )
            DO 50, I = J, N
              A( I, J ) = A( I, J ) + X( I )*TEMP1 + Y( I )*TEMP2
50          CONTINUE
            END IF
60          CONTINUE
          ELSE
            DO 80, J = 1, N
              IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*Y( JY )
                TEMP2 = ALPHA*X( JX )
                IX      = JX
                IY      = JY
                DO 70, I = J, N
                  A( I, J ) = A( I, J ) + X( IX )*TEMP1
$                  + Y( IY )*TEMP2
                  IX      = IX      + INCX
                  IY      = IY      + INCY
70          CONTINUE
                END IF
                JX = JX + INCX
                JY = JY + INCY
80          CONTINUE
            END IF
          END IF
        END IF
*
*      RETURN
*
*      End of DSYR2 .
*
      END

```

— BLAS 2 dsyr2 —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dsyr2 (uplo n alpha x incx y incy a lda)
    (declare (type (simple-array double-float (*)) a y x)
              (type (double-float) alpha)
              (type fixnum lda incy incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%)
       (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 0.0) (temp2 0.0))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                  (type (double-float) temp1 temp2))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((< n 0)
          (setf info 2))
         ((= incx 0)
          (setf info 5))
         ((= incy 0)
          (setf info 7))
         ((< lda (max (the fixnum 1) (the fixnum n)))
          (setf info 9)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DSYR2" info)
          (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
         ((or (/= incx 1) (/= incy 1))
          (cond
           ((> incx 0)
            (setf kx 1))
           (t
            (setf kx
                  (f2cl-lib:int-sub 1

```

```

                                (f2cl-lib:int-mul
                                (f2cl-lib:int-sub n 1)
                                incx))))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
        (f2cl-lib:int-sub 1
                           (f2cl-lib:int-mul
                            (f2cl-lib:int-sub n 1)
                            incy))))))
(setf jx kx)
(setf jy ky))
(cond
  ((char-equal uplo #\U)
   (cond
    ((and (= incx 1) (= incy 1))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
     (tagbody
      (cond
       ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
              (/= (f2cl-lib:fref y (j) ((1 *))) zero))
        (setf temp1
              (* alpha
                 (f2cl-lib:fref y-%data%
                                (j)
                                ((1 *))
                                y-%offset%)))
        (setf temp2
              (* alpha
                 (f2cl-lib:fref x-%data%
                                (j)
                                ((1 *))
                                x-%offset%)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i j) nil)
          (tagbody
           (setf (f2cl-lib:fref a-%data%
                               (i j)
                               ((1 lda) (1 *))
                               a-%offset%)
                  (+
                   (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)
                   (*
                    (f2cl-lib:fref x-%data%

```

```

                                (i)
                                ((1 *))
                                x-%offset%)
                                temp1)
                                (*
                                (f2cl-lib:fref y-%data%
                                (i)
                                ((1 *))
                                y-%offset%)
                                temp2))))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
      (setf temp1
        (* alpha
          (f2cl-lib:fref y-%data%
            (jy)
            ((1 *))
            y-%offset%)))
      (setf temp2
        (* alpha
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)))
      (setf ix kx)
      (setf iy ky)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i j) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
          (+
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
              temp1)
            (*

```

```

(f2cl-lib:fref y-%data%
  (iy)
  ((1 *))
  y-%offset%)
temp2)))
(setf ix (f2cl-lib:int-add ix incx))
(setf iy (f2cl-lib:int-add iy incy))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond
          ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (/= (f2cl-lib:fref y (j) ((1 *))) zero))
            (setf temp1
              (* alpha
                (f2cl-lib:fref y-%data%
                  (j)
                  ((1 *))
                  y-%offset%)))
            (setf temp2
              (* alpha
                (f2cl-lib:fref x-%data%
                  (j)
                  ((1 *))
                  x-%offset%)))
            (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
                (+
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%)
                  (*
                    (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%)
                    temp1)
                  (*
                    (f2cl-lib:fref y-%data%

```

```

                                (i)
                                ((1 *))
                                y-%offset%)
                                temp2))))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
        (setf temp1
          (* alpha
            (f2cl-lib:fref y-%data%
                          (jy)
                          ((1 *))
                          y-%offset%)))

        (setf temp2
          (* alpha
            (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)))

        (setf ix jx)
        (setf iy jy)
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
          (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
              (+
                (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                  temp1)
                (*
                  (f2cl-lib:fref y-%data%
                                (iy)
                                ((1 *))
                                y-%offset%)
                  temp2)))
            (setf ix (f2cl-lib:int-add ix incx))

```



```

                (setf iy (f2cl-lib:int-add iy incy))))))
            (setf jx (f2cl-lib:int-add jx incx))
            (setf jy (f2cl-lib:int-add jy incy))))))
end_label
    (return (values nil nil nil nil nil nil nil nil nil)))

```

dsyr BLAS

— dsyr.input —

```

)set break resume
)sys rm -f dsyr.output
)spool dsyr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dsyr.help —

```

=====
dsyr examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DSYR - perform the symmetric rank 1 operation $A := \alpha * x * x' + A$,

SYNOPSIS

SUBROUTINE DSYR (UPLO, N, ALPHA, X, INCX, A, LDA)

DOUBLE PRECISION ALPHA

INTEGER INCX, LDA, N

CHARACTER*1 UPLO

DOUBLE PRECISION A(LDA, *), X(*)

PURPOSE

DSYR performs the symmetric rank 1 operation

where alpha is a real scalar, x is an n element vector and A is an n by n symmetric matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least

(1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).

Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular

part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

— dsyr.f —

```

SUBROUTINE DSYR ( UPLO, N, ALPHA, X, INCX, A, LDA )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, LDA, N
CHARACTER*1       UPLO
*
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
DOUBLE PRECISION  ZERO
PARAMETER         ( ZERO = 0.0D+0 )
*
* .. Local Scalars ..
DOUBLE PRECISION  TEMP
INTEGER           I, INFO, IX, J, JX, KX
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC         MAX

```

```

*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ).AND.
$          .NOT.LSAME( UPLO, 'L' )          )THEN
          INFO = 1
      ELSE IF( N.LT.0 )THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 5
      ELSE IF( LDA.LT.MAX( 1, N ) )THEN
          INFO = 7
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'DSYR ', INFO )
          RETURN
      END IF

*
*      Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN

*
*      Set the start point in X if the increment is not unity.
*
      IF( INCX.LE.0 )THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
          KX = 1
      END IF

*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through the triangular part
*      of A.
*
      IF( LSAME( UPLO, 'U' ) )THEN

*
*          Form A when A is stored in upper triangle.
*
          IF( INCX.EQ.1 )THEN
              DO 20, J = 1, N
                  IF( X( J ).NE.ZERO )THEN
                      TEMP = ALPHA*X( J )
                      DO 10, I = 1, J
                          A( I, J ) = A( I, J ) + X( I )*TEMP
10                      CONTINUE
                      END IF
                  END IF
              END DO
          END IF
      END IF

```

```

20      CONTINUE
      ELSE
        JX = KX
        DO 40, J = 1, N
          IF( X( JX ).NE.ZERO )THEN
            TEMP = ALPHA*X( JX )
            IX = KX
            DO 30, I = 1, J
              A( I, J ) = A( I, J ) + X( IX )*TEMP
              IX = IX + INCX
            CONTINUE
30          CONTINUE
          END IF
          JX = JX + INCX
40        CONTINUE
      END IF
    ELSE
*
*      Form A when A is stored in lower triangle.
*
      IF( INCX.EQ.1 )THEN
        DO 60, J = 1, N
          IF( X( J ).NE.ZERO )THEN
            TEMP = ALPHA*X( J )
            DO 50, I = J, N
              A( I, J ) = A( I, J ) + X( I )*TEMP
            CONTINUE
50          CONTINUE
          END IF
60        CONTINUE
      ELSE
        JX = KX
        DO 80, J = 1, N
          IF( X( JX ).NE.ZERO )THEN
            TEMP = ALPHA*X( JX )
            IX = JX
            DO 70, I = J, N
              A( I, J ) = A( I, J ) + X( IX )*TEMP
              IX = IX + INCX
            CONTINUE
70          CONTINUE
          END IF
          JX = JX + INCX
80        CONTINUE
      END IF
    END IF
*
    RETURN
*
*      End of DSYR .
*
    END

```

— BLAS 2 dsyr —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dsyr (uplo n alpha x incx a lda)
    (declare (type (simple-array double-float (*)) a x)
              (type (double-float) alpha)
              (type fixnum lda incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x double-float x-%data% x-%offset%)
       (a double-float a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type fixnum i info ix j jx kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((< n 0)
          (setf info 2))
         ((= incx 0)
          (setf info 5))
         ((< lda (max (the fixnum 1) (the fixnum n)))
          (setf info 7)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DSYR" info)
          (go end_label)))
        (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
         ((<= incx 0)
          (setf kx
                (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
         ((/= incx 1)
          (setf kx 1)))
        (cond
         ((char-equal uplo #\U)
          (cond
           ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          (> j n) nil)

```

```

(tagbody
  (cond
    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref x-%data%
            (j)
            ((1 *))
            x-%offset%)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i j) nil)
        (tagbody
          (setf (f2cl-lib:fref a-%data%
            (i j)
            ((1 lda) (1 *))
            a-%offset%)
            (+
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
              (*
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%)
                temp))))))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (* alpha
              (f2cl-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)))
          (setf ix kx)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i j) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
                (+
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%)
                  (*
                    (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%)
                    temp))))))))))

```

```

                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                                (*
                                (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                                temp)))
                                (setf ix (f2cl-lib:int-add ix incx))))))
                                (setf jx (f2cl-lib:int-add jx incx))))))
(t
(cond
  ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
          (setf temp
            (* alpha
              (f2cl-lib:fref x-%data%
                (j)
                ((1 *))
                x-%offset%)))
          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
            (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
              (+
                (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%)
                  temp))))))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)

```



```

      (setf temp
        (* alpha
          (f2cl-lib:fref x-%data%
                        (jx)
                        ((1 *))
                        x-%offset%)))

      (setf ix jx)
      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                    (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
          (+
            (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                            (ix)
                            ((1 *))
                            x-%offset%)
              temp)))
        (setf ix (f2cl-lib:int-add ix incx))))))
      (setf jx (f2cl-lib:int-add jx incx))))))
end_label
(return (values nil nil nil nil nil nil nil))))

```

dtbmvm BLAS

— dtbmvm.input —

```

)set break resume
)sys rm -f dtbmvm.output
)spool dtbmvm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtbm.v.help —

```
=====
dtbm.v examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DTBMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$,

SYNOPSIS

```
SUBROUTINE DTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                  )
```

```
INTEGER          INCX, K, LDA, N
```

```
CHARACTER*1      DIAG, TRANS, UPLO
```

```
DOUBLE           PRECISION A( LDA, * ), X( * )
```

PURPOSE

DTBMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := A*x$.

TRANS = 'T' or 't' x := A'*x.

TRANS = 'C' or 'c' x := A'*x.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 ≤ K. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N
  M = K + 1 - J
  DO 10, I = MAX( 1, J - K ), J
    A( M + I, J ) = matrix( I, J )
  10 CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal

of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
) A( M + I, J ) = matrix( I, J ) 10    CONTINUE 20
CONTINUE
```

Note that when $DIAG = 'U'$ or $'u'$ the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $(k + 1)$. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x . On exit, X is overwritten with the transformed vector x .

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X . INCX must not be zero. Unchanged on exit.

— dtbm.f —

```
SUBROUTINE DTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
* .. Scalar Arguments ..
INTEGER          INCX, K, LDA, N
CHARACTER*1      DIAG, TRANS, UPLO
* .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), X( * )
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
```

```

*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.
*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
      DOUBLE PRECISION    ZERO
      PARAMETER            ( ZERO = 0.0D+0 )
*
*      .. Local Scalars ..
      DOUBLE PRECISION    TEMP
      INTEGER              I, INFO, IX, J, JX, KPLUS1, KX, L
      LOGICAL              NOUNIT
*
*      .. External Functions ..
      LOGICAL              LSAME
      EXTERNAL             LSAME
*
*      .. External Subroutines ..
      EXTERNAL             XERBLA
*
*      .. Intrinsic Functions ..
      INTRINSIC            MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ) ).AND.
$           .NOT.LSAME( UPLO , 'L' ) ) THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ) ).AND.
$           .NOT.LSAME( TRANS, 'T' ) ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ) ).AND.
$           .NOT.LSAME( DIAG , 'N' ) ) THEN
          INFO = 3
      ELSE IF( N.LT.0 ) THEN
          INFO = 4
      ELSE IF( K.LT.0 ) THEN
          INFO = 5
      ELSE IF( LDA.LT.( K + 1 ) ) THEN
          INFO = 7
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 9
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DTBMV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.

```

```

*
  IF( N.EQ.0 )
$   RETURN
*
  NOUNIT = LSAME( DIAG, 'N' )
*
*   Set up the start point in X if the increment is not unity. This
*   will be ( N - 1 )*INCX too small for descending loops.
*
  IF( INCX.LE.0 )THEN
    KX = 1 - ( N - 1 )*INCX
  ELSE IF( INCX.NE.1 )THEN
    KX = 1
  END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through A.
*
  IF( LSAME( TRANS, 'N' ) )THEN
*
*     Form x := A*x.
*
    IF( LSAME( UPLO, 'U' ) )THEN
      KPLUS1 = K + 1
      IF( INCX.EQ.1 )THEN
        DO 20, J = 1, N
          IF( X( J ).NE.ZERO )THEN
            TEMP = X( J )
            L = KPLUS1 - J
            DO 10, I = MAX( 1, J - K ), J - 1
              X( I ) = X( I ) + TEMP*A( L + I, J )
10          CONTINUE
            IF( NOUNIT )
$              X( J ) = X( J )*A( KPLUS1, J )
            END IF
          20 CONTINUE
        ELSE
          JX = KX
          DO 40, J = 1, N
            IF( X( JX ).NE.ZERO )THEN
              TEMP = X( JX )
              IX = KX
              L = KPLUS1 - J
              DO 30, I = MAX( 1, J - K ), J - 1
                X( IX ) = X( IX ) + TEMP*A( L + I, J )
                IX = IX + INCX
30              CONTINUE
              IF( NOUNIT )
$                X( JX ) = X( JX )*A( KPLUS1, J )
            END IF
          40 CONTINUE
        END IF
      END IF
    END IF
  END IF

```

```

        JX = JX + INCX
        IF( J.GT.K )
$           KX = KX + INCX
40      CONTINUE
      END IF
    ELSE
      IF( INCX.EQ.1 )THEN
        DO 60, J = N, 1, -1
          IF( X( J ).NE.ZERO )THEN
            TEMP = X( J )
            L     = 1          - J
            DO 50, I = MIN( N, J + K ), J + 1, -1
              X( I ) = X( I ) + TEMP*A( L + I, J )
50          CONTINUE
            IF( NOUNIT )
$              X( J ) = X( J )*A( 1, J )
            END IF
60        CONTINUE
      ELSE
        KX = KX + ( N - 1 )*INCX
        JX = KX
        DO 80, J = N, 1, -1
          IF( X( JX ).NE.ZERO )THEN
            TEMP = X( JX )
            IX    = KX
            L     = 1          - J
            DO 70, I = MIN( N, J + K ), J + 1, -1
              X( IX ) = X( IX ) + TEMP*A( L + I, J )
              IX      = IX      - INCX
70          CONTINUE
            IF( NOUNIT )
$              X( JX ) = X( JX )*A( 1, J )
            END IF
            JX = JX - INCX
            IF( ( N - J ).GE.K )
$              KX = KX - INCX
80          CONTINUE
        END IF
      END IF
    ELSE
*
*      Form  x := A'*x.
*
      IF( LSAME( UPLO, 'U' ) )THEN
        KPLUS1 = K + 1
        IF( INCX.EQ.1 )THEN
          DO 100, J = N, 1, -1
            TEMP = X( J )
            L     = KPLUS1 - J
            IF( NOUNIT )

```

```

$          TEMP = TEMP*A( KPLUS1, J )
          DO 90, I = J - 1, MAX( 1, J - K ), -1
              TEMP = TEMP + A( L + I, J )*X( I )
90          CONTINUE
          X( J ) = TEMP
100         CONTINUE
        ELSE
          KX = KX + ( N - 1 )*INCX
          JX = KX
          DO 120, J = N, 1, -1
              TEMP = X( JX )
              KX = KX - INCX
              IX = KX
              L = KPLUS1 - J
              IF( NUNIT )
$              TEMP = TEMP*A( KPLUS1, J )
              DO 110, I = J - 1, MAX( 1, J - K ), -1
                  TEMP = TEMP + A( L + I, J )*X( IX )
                  IX = IX - INCX
110             CONTINUE
              X( JX ) = TEMP
              JX = JX - INCX
120             CONTINUE
          END IF
        ELSE
          IF( INCX.EQ.1 )THEN
            DO 140, J = 1, N
              TEMP = X( J )
              L = 1 - J
              IF( NUNIT )
$              TEMP = TEMP*A( 1, J )
              DO 130, I = J + 1, MIN( N, J + K )
                  TEMP = TEMP + A( L + I, J )*X( I )
130             CONTINUE
              X( J ) = TEMP
140             CONTINUE
            ELSE
              JX = KX
              DO 160, J = 1, N
                  TEMP = X( JX )
                  KX = KX + INCX
                  IX = KX
                  L = 1 - J
                  IF( NUNIT )
$                  TEMP = TEMP*A( 1, J )
                  DO 150, I = J + 1, MIN( N, J + K )
                      TEMP = TEMP + A( L + I, J )*X( IX )
                      IX = IX + INCX
150                  CONTINUE
                  X( JX ) = TEMP

```



```

          JX      = JX  + INCX
160      CONTINUE
          END IF
      END IF
  END IF
*
  RETURN
*
*   End of DTBMV .
*
  END

```

— BLAS 2 dtbm —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtbm (uplo trans diag n k a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
              (type fixnum incx lda k n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kplus1 0) (kx 0)
             (l 0) (temp 0.0))
        (declare (type (member t nil) nounit)
                  (type fixnum i info ix j jx kplus1 kx l)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
          (setf info 2))
         ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
          (setf info 3))
         ((< n 0)
          (setf info 4))
         ((< k 0)
          (setf info 5))
         ((< lda (f2cl-lib:int-add k 1))

```

```

    (setf info 7))
    (= incx 0)
    (setf info 9)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DTBMV" info)
  (go end_label)))
(if (= n 0) (go end_label))
(setf nunit (char-equal diag #\N))
(cond
  (<= incx 0)
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx))))

  (/= incx 1)
  (setf kx 1)))
(cond
  (char-equal trans #\N)
  (cond
    (char-equal uplo #\U)
    (setf kplus1 (f2cl-lib:int-add k 1))
    (cond
      (= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond
          (/= (f2cl-lib:fref x (j) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))
              (f2cl-lib:int-add i 1))
            (> i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub
                  1))))
            nil)
          (tagbody
            (setf (f2cl-lib:fref x-%data%
              (i)
              ((1 *))

```

```

                                x-%offset%)
(+
  (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
  (* temp
    (f2cl-lib:fref a-%data%
      ((f2cl-lib:int-add 1 i)
      j)
      ((1 lda) (1 *))
      a-%offset%))))))
(if nunit
  (setf (f2cl-lib:fref x-%data%
    (j)
    ((1 *))
    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
        (j)
        ((1 *))
        x-%offset%)
      (f2cl-lib:fref a-%data%
        (kplus1 j)
        ((1 lda) (1 *))
        a-%offset%)))))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%))
          (setf ix kx)
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))
            (f2cl-lib:int-add i 1))
            ((> i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub

```

```

1)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
        (+
          (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
          (* temp
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i)
                           j)
                          ((1 lda) (1 *))
                          a-%offset%)))
          (setf ix (f2cl-lib:int-add ix incx))))
      (if nount
        (setf (f2cl-lib:fref x-%data%
                            (jx)
                            ((1 *))
                            x-%offset%)
          (*
            (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
            (f2cl-lib:fref a-%data%
                          (kplus1 j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
        (setf jx (f2cl-lib:int-add jx incx))
        (if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))
  (t
   (cond
    ((= incx 1)
     (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                   ((> j 1) nil)
     (tagbody
      (cond
       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf l (f2cl-lib:int-sub 1 j))
        (f2cl-lib:fdo (i
          (min (the fixnum n)
               (the fixnum
                 (f2cl-lib:int-add j k)))

```

```

(f2cl-lib:int-add i
  (f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1) nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
    (+
      (f2cl-lib:fref x-%data%
        (i)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i)
           j)
          ((1 lda) (1 *))
          a-%offset%))))))
(if nunit
  (setf (f2cl-lib:fref x-%data%
    (j)
    ((1 *))
    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
        (j)
        ((1 *))
        x-%offset%)
      (f2cl-lib:fref a-%data%
        (1 j)
        ((1 lda) (1 *))
        a-%offset%))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%))

```

```

(setf ix kx)
(setf l (f2cl-lib:int-sub 1 j))
(f2cl-lib:fdo (i
              (min (the fixnum n)
                    (the fixnum
                      (f2cl-lib:int-add j k)))
              (f2cl-lib:int-add i
                                (f2cl-lib:int-sub 1)))
              ((> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
          (+
            (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                            ((f2cl-lib:int-add 1 i)
                             j)
                            ((1 lda) (1 *))
                            a-%offset%))))
    (setf ix (f2cl-lib:int-sub ix incx)))
  (if nounit
    (setf (f2cl-lib:fref x-%data%
                        (jx)
                        ((1 *))
                        x-%offset%)
          (*
            (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
            (f2cl-lib:fref a-%data%
                          (1 j)
                          ((1 lda) (1 *))
                          a-%offset%))))
    (setf jx (f2cl-lib:int-sub jx incx))
    (if (>= (f2cl-lib:int-sub n j) k)
      (setf kx (f2cl-lib:int-sub kx incx))))))
(t
 (cond
  ((char-equal uplo #\U)
   (setf kplus1 (f2cl-lib:int-add k 1))
   (cond
    ((= incx 1)
     (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))

```

```

        (> j 1) nil)
(tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (setf l (f2cl-lib:int-sub kplus1 j))
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:fref a-%data%
          (kplus1 j)
          ((1 lda) (1 *))
          a-%offset%))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
      (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
      (> i
        (max (the fixnum 1)
          (the fixnum
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub
                k))))))
      nil)
  (tagbody
    (setf temp
      (+ temp
        (*
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add 1 i) j)
            ((1 lda) (1 *))
            a-%offset%)
          (f2cl-lib:fref x-%data%
            (i)
            ((1 *))
            x-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf kx (f2cl-lib:int-sub kx incx))
    (setf ix kx)
    (setf l (f2cl-lib:int-sub kplus1 j))

```

```

(if nounit
  (setf temp
    (* temp
      (f2cl-lib:fref a-%data%
        (kplus1 j)
        ((1 lda) (1 *))
        a-%offset%))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
  (> i
    (max (the fixnum 1)
      (the fixnum
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub
            k))))))
  nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i) j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))
      (setf ix (f2cl-lib:int-sub ix incx))))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-sub jx incx))))))

(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *) x-%offset%))
          (setf 1 (f2cl-lib:int-sub 1 j))
          (if nounit
            (setf temp
              (* temp
                (f2cl-lib:fref a-%data%
                  (1 j)
                  ((1 lda) (1 *))
                  a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
              (f2cl-lib:int-add i 1))

```



```

        (> i
         (min (the fixnum n)
              (the fixnum
                (f2cl-lib:int-add j k))))
        nil)
    (tagbody
     (setf temp
       (+ temp
          (*
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i) j)
                          ((1 lda) (1 *)))
            a-%offset%)
            (f2cl-lib:fref x-%data%
                          (i)
                          ((1 *))
                          x-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
          temp)))
(t
 (setf jx kx)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               (> j n) nil)
 (tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
  (setf kx (f2cl-lib:int-add kx incx))
  (setf ix kx)
  (setf l (f2cl-lib:int-sub 1 j))
  (if nunit
    (setf temp
      (* temp
         (f2cl-lib:fref a-%data%
                       (1 j)
                       ((1 lda) (1 *)))
         a-%offset%)))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
                  (> i
                   (min (the fixnum n)
                        (the fixnum
                          (f2cl-lib:int-add j k))))
                  nil)
    (tagbody
     (setf temp
       (+ temp
          (*
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i) j)
                          ((1 lda) (1 *)))

```

```

                                a-%offset%)
(f2cl-lib:fref x-%data%
  (ix)
  ((1 *))
  x-%offset%))))
  (setf ix (f2cl-lib:int-add ix incx)))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-add jx incx)))))))))
end_label
  (return (values nil nil nil nil nil nil nil nil)))

```

dtbsv BLAS

— dtbsv.input —

```

)set break resume
)sys rm -f dtbsv.output
)spool dtbsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtbsv.help —

```

=====
dtbsv examples
=====

=====
Man Page Details
=====

NAME
    DTBSV - solve one of the systems of equations  $A*x = b$ , or
     $A'*x = b$ ,

SYNOPSIS

```

```
SUBROUTINE DTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                  )
```

```
INTEGER          INCX, K, LDA, N
```

```
CHARACTER*1      DIAG, TRANS, UPLO
```

```
DOUBLE           PRECISION A( LDA, * ), X( * )
```

PURPOSE

DTBSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $A*x = b$.

TRANS = 'T' or 't' $A'*x = b$.

TRANS = 'C' or 'c' $A'*x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
K ), J A( M + I, J ) = matrix( I, J ) 10        CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
) A( M + I, J ) = matrix( I, J ) 10        CONTINUE 20
CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the

matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least (1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— dtbsv.f —

```

SUBROUTINE DTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
*   .. Scalar Arguments ..
    INTEGER          INCX, K, LDA, N
    CHARACTER*1      DIAG, TRANS, UPLO
*   .. Array Arguments ..
    DOUBLE PRECISION A( LDA, * ), X( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*       Jack Dongarra, Argonne National Lab.
*       Jeremy Du Croz, Nag Central Office.
*       Sven Hammarling, Nag Central Office.
*       Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
    DOUBLE PRECISION ZERO
    PARAMETER      ( ZERO = 0.0D+0 )
*   .. Local Scalars ..
    DOUBLE PRECISION TEMP
    INTEGER        I, INFO, IX, J, JX, KPLUS1, KX, L
    LOGICAL        NOUNIT
*   .. External Functions ..

```

```

LOGICAL          LSAME
EXTERNAL         LSAME
* .. External Subroutines ..
EXTERNAL         XERBLA
* .. Intrinsic Functions ..
INTRINSIC        MAX, MIN
*
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      IF ( .NOT.LSAME( UPLO , 'U' ).AND.
$        .NOT.LSAME( UPLO , 'L' ) ) THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$          .NOT.LSAME( TRANS, 'T' ).AND.
$          .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$          .NOT.LSAME( DIAG , 'N' ) ) THEN
          INFO = 3
      ELSE IF( N.LT.0 ) THEN
          INFO = 4
      ELSE IF( K.LT.0 ) THEN
          INFO = 5
      ELSE IF( LDA.LT.( K + 1 ) ) THEN
          INFO = 7
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 9
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DTBSV ', INFO )
          RETURN
      END IF
*
* Quick return if possible.
*
      IF( N.EQ.0 )
$        RETURN
*
      NOUNIT = LSAME( DIAG, 'N' )
*
* Set up the start point in X if the increment is not unity. This
* will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 ) THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 ) THEN
          KX = 1

```

```

      END IF
*
*   Start the operations. In this version the elements of A are
*   accessed by sequentially with one pass through A.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*       Form  $x := inv(A)x$ .
*
      IF( LSAME( UPLO, 'U' ) )THEN
        KPLUS1 = K + 1
        IF( INCX.EQ.1 )THEN
          DO 20, J = N, 1, -1
            IF( X( J ).NE.ZERO )THEN
              L = KPLUS1 - J
              IF( NOUNIT )
$                X( J ) = X( J )/A( KPLUS1, J )
              TEMP = X( J )
              DO 10, I = J - 1, MAX( 1, J - K ), -1
                X( I ) = X( I ) - TEMP*A( L + I, J )
10              CONTINUE
            END IF
          20        CONTINUE
        ELSE
          KX = KX + ( N - 1 )*INCX
          JX = KX
          DO 40, J = N, 1, -1
            KX = KX - INCX
            IF( X( JX ).NE.ZERO )THEN
              IX = KX
              L = KPLUS1 - J
              IF( NOUNIT )
$                X( JX ) = X( JX )/A( KPLUS1, J )
              TEMP = X( JX )
              DO 30, I = J - 1, MAX( 1, J - K ), -1
                X( IX ) = X( IX ) - TEMP*A( L + I, J )
                IX = IX - INCX
30              CONTINUE
            END IF
            JX = JX - INCX
          40        CONTINUE
        END IF
      ELSE
        IF( INCX.EQ.1 )THEN
          DO 60, J = 1, N
            IF( X( J ).NE.ZERO )THEN
              L = 1 - J
              IF( NOUNIT )
$                X( J ) = X( J )/A( 1, J )
              TEMP = X( J )

```

```

DO 50, I = J + 1, MIN( N, J + K )
    X( I ) = X( I ) - TEMP*A( L + I, J )
50    CONTINUE
    END IF
60    CONTINUE
ELSE
    JX = KX
    DO 80, J = 1, N
        KX = KX + INCX
        IF( X( JX ).NE.ZERO )THEN
            IX = KX
            L = 1 - J
            IF( NOUNIT )
                $    X( JX ) = X( JX )/A( 1, J )
                TEMP = X( JX )
            DO 70, I = J + 1, MIN( N, J + K )
                X( IX ) = X( IX ) - TEMP*A( L + I, J )
                IX = IX + INCX
70        CONTINUE
            END IF
            JX = JX + INCX
80        CONTINUE
        END IF
    END IF
ELSE
*
*    Form x := inv( A' )*x.
*
    IF( LSAME( UPLO, 'U' ) )THEN
        KPLUS1 = K + 1
        IF( INCX.EQ.1 )THEN
            DO 100, J = 1, N
                TEMP = X( J )
                L = KPLUS1 - J
                DO 90, I = MAX( 1, J - K ), J - 1
                    TEMP = TEMP - A( L + I, J )*X( I )
90        CONTINUE
                IF( NOUNIT )
                    $    TEMP = TEMP/A( KPLUS1, J )
                X( J ) = TEMP
100       CONTINUE
            ELSE
                JX = KX
                DO 120, J = 1, N
                    TEMP = X( JX )
                    IX = KX
                    L = KPLUS1 - J
                    DO 110, I = MAX( 1, J - K ), J - 1
                        TEMP = TEMP - A( L + I, J )*X( IX )
                        IX = IX + INCX

```



```

110          CONTINUE
            IF( NOUNIT )
$              TEMP = TEMP/A( KPLUS1, J )
              X( JX ) = TEMP
              JX      = JX  + INCX
              IF( J.GT.K )
$                KX = KX + INCX
120          CONTINUE
            END IF
          ELSE
            IF( INCX.EQ.1 )THEN
              DO 140, J = N, 1, -1
                TEMP = X( J )
                L     = 1      - J
                DO 130, I = MIN( N, J + K ), J + 1, -1
                  TEMP = TEMP - A( L + I, J )*X( I )
130              CONTINUE
                IF( NOUNIT )
$                  TEMP = TEMP/A( 1, J )
                X( J ) = TEMP
140              CONTINUE
            ELSE
              KX = KX + ( N - 1 )*INCX
              JX = KX
              DO 160, J = N, 1, -1
                TEMP = X( JX )
                IX   = KX
                L     = 1      - J
                DO 150, I = MIN( N, J + K ), J + 1, -1
                  TEMP = TEMP - A( L + I, J )*X( IX )
                  IX   = IX  - INCX
150              CONTINUE
                IF( NOUNIT )
$                  TEMP = TEMP/A( 1, J )
                X( JX ) = TEMP
                JX      = JX  - INCX
                IF( ( N - J ).GE.K )
$                  KX = KX - INCX
160              CONTINUE
            END IF
          END IF
        END IF
      *
      RETURN
    *
    *   End of DTBSV .
    *
    END

```

— BLAS 2 dtbsv —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtbsv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
              (type fixnum incx lda k n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nunit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kplus1 0) (kx 0)
             (l 0) (temp 0.0))
        (declare (type (member t nil) nunit)
                  (type fixnum i info ix j jx kplus1 kx l)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
          (setf info 2))
         ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
          (setf info 3))
         ((< n 0)
          (setf info 4))
         ((< k 0)
          (setf info 5))
         ((< lda (f2cl-lib:int-add k 1))
          (setf info 7))
         ((= incx 0)
          (setf info 9)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DTBSV" info)
          (go end_label)))
        (if (= n 0) (go end_label))
        (setf nunit (char-equal diag #\N))
        (cond
         ((<= incx 0)

```

```

(setf kx
  (f2cl-lib:int-sub 1
    (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
      incx))))

( (/ = incx 1)
  (setf kx 1)))
(cond
  ((char-equal trans #\N)
    (cond
      ((char-equal uplo #\U)
        (setf kplus1 (f2cl-lib:int-add k 1))
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j 1) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                  (setf l (f2cl-lib:int-sub kplus1 j))
                  (if nunit
                    (setf (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                      (/
                        (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                        (f2cl-lib:fref a-%data%
                                          (kplus1 j)
                                          ((1 lda) (1 *))
                                          a-%offset%))))))
                  (setf temp
                    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (f2cl-lib:fdo (i
                    (f2cl-lib:int-add j
                      (f2cl-lib:int-sub 1))
                    (f2cl-lib:int-add i
                      (f2cl-lib:int-sub 1)))
                      ((> i
                        (max (the fixnum 1)
                          (the fixnum
                            (f2cl-lib:int-add j
                              (f2cl-lib:int-sub
                                k))))))
                        nil)
                    (tagbody
                      (setf (f2cl-lib:fref x-%data%
                                            (i)

```

```

((1 *))
x-%offset%)
(-
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
(* temp
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add 1 i)
j)
((1 lda) (1 *))
a-%offset%)))))))))
(t
(setf kx
(f2cl-lib:int-add kx
(f2cl-lib:int-mul
(f2cl-lib:int-sub n 1)
incx)))
(setf jx kx)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
(> j 1) nil)
(tagbody
(setf kx (f2cl-lib:int-sub kx incx))
(cond
((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
(setf ix kx)
(setf l (f2cl-lib:int-sub kplus1 j))
(if nounit
(setf (f2cl-lib:fref x-%data%
(jx)
((1 *))
x-%offset%)
(/
(f2cl-lib:fref x-%data%
(jx)
((1 *))
x-%offset%)
(f2cl-lib:fref a-%data%
(kplus1 j)
((1 lda) (1 *))
a-%offset%))))))
(setf temp
(f2cl-lib:fref x-%data%
(jx)
((1 *))
x-%offset%))
(f2cl-lib:fdo (i
(f2cl-lib:int-add j
(f2cl-lib:int-sub 1))

```

```

        (f2cl-lib:int-add i
          (f2cl-lib:int-sub 1)))
      (> i
        (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))))
      nil)
    (tagbody
      (setf (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
        (-
          (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
          (* temp
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i)
                           j)
                          ((1 lda) (1 *))
                          a-%offset%))))
        (setf ix (f2cl-lib:int-sub ix incx))))))
      (setf jx (f2cl-lib:int-sub jx incx))))))
  (t
   (cond
    ((= incx 1)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (cond
       ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf 1 (f2cl-lib:int-sub 1 j))
        (if nunit
         (setf (f2cl-lib:fref x-%data%
                              (j)
                              ((1 *))
                              x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                              (j)
                              ((1 *))
                              x-%offset%)
                (f2cl-lib:fref a-%data%
                              (1 j)
                              ((1 lda) (1 *))
                              a-%offset%))))
        ))
      ))
    ))

```



```

(f2cl-lib:fref x-%data%
  (jx)
  ((1 *))
  x-%offset%))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i
    (min (the fixnum n)
      (the fixnum
        (f2cl-lib:int-add j k))))
    nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
    (ix)
    ((1 *))
    x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i)
            j)
            ((1 lda) (1 *))
            a-%offset%))))
      (setf ix (f2cl-lib:int-add ix incx))))))
(setf jx (f2cl-lib:int-add jx incx))))))
(t
  (cond
    ((char-equal uplo #\U)
      (setf kplus1 (f2cl-lib:int-add k 1))
      (cond
        ((= incx 1)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            (> j n) nil)
          (tagbody
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (setf l (f2cl-lib:int-sub kplus1 j))
            (f2cl-lib:fdo (i
              (max (the fixnum 1)
                (the fixnum
                  (f2cl-lib:int-add j
                    (f2cl-lib:int-sub
                      k))))
                (f2cl-lib:int-add i 1))
              (> i
                (f2cl-lib:int-add j

```

```

(f2cl-lib:int-sub 1)))

nil)

(tagbody
  (setf temp
    (- temp
      (*
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i) j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))

(if nounit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (kplus1 j)
        ((1 lda) (1 *))
        a-%offset%)))
  (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
    temp))))

(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix kx)
      (setf l (f2cl-lib:int-sub kplus1 j))
      (f2cl-lib:fdo (i
        (max (the fixnum 1)
          (the fixnum
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub
                k))))
          (f2cl-lib:int-add i 1))
        ((> i
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub 1)))
          nil)
        (tagbody
          (setf temp
            (- temp
              (*
                (f2cl-lib:fref a-%data%
                  ((f2cl-lib:int-add 1 i) j)
                  ((1 lda) (1 *))

```



```

                                a-%offset%)
                                (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%))))
                                (setf ix (f2cl-lib:int-add ix incx))))
(if nounit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (kplus1 j)
        ((1 lda) (1 *))
        a-%offset%))))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-add jx incx))
  (if (> j k) (setf kx (f2cl-lib:int-add kx incx)))))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        ((> j 1) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf l (f2cl-lib:int-sub 1 j))
        (f2cl-lib:fdo (i
          (min (the fixnum n)
            (the fixnum
              (f2cl-lib:int-add j k)))
          (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
          ((> i (f2cl-lib:int-add j 1)) nil)
        (tagbody
          (setf temp
            (- temp
              (*
                (f2cl-lib:fref a-%data%
                  ((f2cl-lib:int-add 1 i) j)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%))))))
          (if nounit
            (setf temp
              (/ temp
                (f2cl-lib:fref a-%data%
                  (1 j)
                  ((1 lda) (1 *))

```

```

                                a-%offset%)))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
          temp)))
(t
  (setf kx
        (f2cl-lib:int-add kx
                          (f2cl-lib:int-mul
                          (f2cl-lib:int-sub n 1)
                          incx)))
    (setf jx kx)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                  (> j 1) nil)
    (tagbody
      (setf temp
              (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix kx)
      (setf l (f2cl-lib:int-sub 1 j))
      (f2cl-lib:fdo (i
                    (min (the fixnum n)
                        (the fixnum
                          (f2cl-lib:int-add j k)))
                    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                    (> i (f2cl-lib:int-add j 1)) nil)
        (tagbody
          (setf temp
                  (- temp
                     (*
                      (f2cl-lib:fref a-%data%
                                      ((f2cl-lib:int-add l i) j)
                                      ((1 lda) (1 *))
                                      a-%offset%)
                      (f2cl-lib:fref x-%data%
                                      (ix)
                                      ((1 *))
                                      x-%offset%))))))
          (setf ix (f2cl-lib:int-sub ix incx)))
        (if nounit
            (setf temp
                    (/ temp
                       (f2cl-lib:fref a-%data%
                                       (1 j)
                                       ((1 lda) (1 *))
                                       a-%offset%))))
            (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                  temp)
            (setf jx (f2cl-lib:int-sub jx incx))
            (if (>= (f2cl-lib:int-sub n j) k)
                (setf kx (f2cl-lib:int-sub kx incx))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

dtpmv BLAS

— dtpmv.input —

```
)set break resume
)sys rm -f dtpmv.output
)spool dtpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dtpmv.help —

```
=====
dtpmv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DTPMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$,

SYNOPSIS

SUBROUTINE DTPMV (UPLO, TRANS, DIAG, N, AP, X, INCX)

INTEGER INCX, N

CHARACTER*1 DIAG, TRANS, UPLO

DOUBLE PRECISION AP(*), X(*)

PURPOSE

DTPMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := Ax$.

TRANS = 'T' or 't' $x := A^T x$.

TRANS = 'C' or 'c' $x := A^H x$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A . N must be at least zero. Unchanged on exit.

AP - DOUBLE PRECISION array of DIMENSION at least $((n*(n+1))/2)$. Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains $a(1, 1)$, AP(2) and AP(3) contain $a(1, 2)$ and $a(2, 2)$ respectively, and so

on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least (1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

—————

— dtpmv.f —

```

SUBROUTINE DTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )
*
* .. Scalar Arguments ..
  INTEGER          INCX, N
  CHARACTER*1      DIAG, TRANS, UPLO
*
* .. Array Arguments ..
  DOUBLE PRECISION AP( * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
  DOUBLE PRECISION ZERO
  PARAMETER      ( ZERO = 0.0D+0 )
*
* .. Local Scalars ..
  DOUBLE PRECISION TEMP
  INTEGER         I, INFO, IX, J, JX, K, KK, KX
  LOGICAL         NOUNIT

```

```

*      .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*      .. External Subroutines ..
      EXTERNAL         XERBLA
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$           .NOT.LSAME( UPLO , 'L' )      )THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$           .NOT.LSAME( TRANS, 'T' ).AND.
$           .NOT.LSAME( TRANS, 'C' )      )THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$           .NOT.LSAME( DIAG , 'N' )      )THEN
          INFO = 3
      ELSE IF( N.LT.0 )THEN
          INFO = 4
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 7
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'DTPMV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
      NOUNIT = LSAME( DIAG, 'N' )
*
*      Set up the start point in X if the increment is not unity. This
*      will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
          KX = 1
      END IF
*
*      Start the operations. In this version the elements of AP are
*      accessed sequentially with one pass through AP.
*

```

```

      IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form  x:= A*x.
*
      IF( LSAME( UPLO, 'U' ) )THEN
        KK =1
        IF( INCX.EQ.1 )THEN
          DO 20, J = 1, N
            IF( X( J ).NE.ZERO )THEN
              TEMP = X( J )
              K      = KK
              DO 10, I = 1, J - 1
                X( I ) = X( I ) + TEMP*AP( K )
                K      = K      + 1
10             CONTINUE
              IF( NUNIT )
                $      X( J ) = X( J )*AP( KK + J - 1 )
              END IF
              KK = KK + J
20             CONTINUE
            ELSE
              JX = KX
              DO 40, J = 1, N
                IF( X( JX ).NE.ZERO )THEN
                  TEMP = X( JX )
                  IX   = KX
                  DO 30, K = KK, KK + J - 2
                    X( IX ) = X( IX ) + TEMP*AP( K )
                    IX      = IX      + INCX
30                 CONTINUE
                  IF( NUNIT )
                    $      X( JX ) = X( JX )*AP( KK + J - 1 )
                  END IF
                  JX = JX + INCX
                  KK = KK + J
40                 CONTINUE
                END IF
              ELSE
                KK = ( N*( N + 1 ) )/2
                IF( INCX.EQ.1 )THEN
                  DO 60, J = N, 1, -1
                    IF( X( J ).NE.ZERO )THEN
                      TEMP = X( J )
                      K      = KK
                      DO 50, I = N, J + 1, -1
                        X( I ) = X( I ) + TEMP*AP( K )
                        K      = K      - 1
50                     CONTINUE
                      IF( NUNIT )
                        $      X( J ) = X( J )*AP( KK - N + J )

```

```

        END IF
        KK = KK - ( N - J + 1 )
60      CONTINUE
      ELSE
        KX = KX + ( N - 1 )*INCX
        JX = KX
        DO 80, J = N, 1, -1
          IF( X( JX ).NE.ZERO )THEN
            TEMP = X( JX )
            IX = KX
            DO 70, K = KK, KK - ( N - ( J + 1 ) ), -1
              X( IX ) = X( IX ) + TEMP*AP( K )
              IX = IX - INCX
70          CONTINUE
            IF( NOUNIT )
              $      X( JX ) = X( JX )*AP( KK - N + J )
            END IF
            JX = JX - INCX
            KK = KK - ( N - J + 1 )
80      CONTINUE
        END IF
      END IF
    ELSE
      *
      *      Form  x := A'*x.
      *
      IF( LSAME( UPLO, 'U' ) )THEN
        KK = ( N*( N + 1 ) )/2
        IF( INCX.EQ.1 )THEN
          DO 100, J = N, 1, -1
            TEMP = X( J )
            IF( NOUNIT )
              $      TEMP = TEMP*AP( KK )
            K = KK - 1
            DO 90, I = J - 1, 1, -1
              TEMP = TEMP + AP( K )*X( I )
              K = K - 1
90          CONTINUE
            X( J ) = TEMP
            KK = KK - J
100         CONTINUE
          ELSE
            JX = KX + ( N - 1 )*INCX
            DO 120, J = N, 1, -1
              TEMP = X( JX )
              IX = JX
              IF( NOUNIT )
                $      TEMP = TEMP*AP( KK )
              DO 110, K = KK - 1, KK - J + 1, -1
                IX = IX - INCX

```



```

        TEMP = TEMP + AP( K ) * X( IX )
110      CONTINUE
        X( JX ) = TEMP
        JX      = JX  - INCX
        KK      = KK  - J
120      CONTINUE
      END IF
    ELSE
      KK = 1
      IF( INCX.EQ.1 ) THEN
        DO 140, J = 1, N
          TEMP = X( J )
          IF( NUNIT )
$            TEMP = TEMP * AP( KK )
            K = KK + 1
            DO 130, I = J + 1, N
              TEMP = TEMP + AP( K ) * X( I )
              K = K + 1
130          CONTINUE
          X( J ) = TEMP
          KK = KK + ( N - J + 1 )
140        CONTINUE
      ELSE
        JX = KX
        DO 160, J = 1, N
          TEMP = X( JX )
          IX = JX
          IF( NUNIT )
$            TEMP = TEMP * AP( KK )
            DO 150, K = KK + 1, KK + N - J
              IX = IX + INCX
              TEMP = TEMP + AP( K ) * X( IX )
150          CONTINUE
          X( JX ) = TEMP
          JX = JX + INCX
          KK = KK + ( N - J + 1 )
160        CONTINUE
      END IF
    END IF
  END IF
*
  RETURN
*
*   End of DTPMV .
*
  END

```

— BLAS 2 dtpmv —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtpmv (uplo trans diag n ap x incx)
    (declare (type (simple-array double-float (*)) x ap)
              (type fixnum incx n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (ap double-float ap-%data% ap-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nunit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0)
              (kx 0) (temp 0.0))
        (declare (type (member t nil) nunit)
                  (type fixnum i info ix j jx k kk kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
          (setf info 2))
         ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
          (setf info 3))
         ((< n 0)
          (setf info 4))
         ((= incx 0)
          (setf info 7)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DTPMV" info)
          (go end_label)))
        (if (= n 0) (go end_label))
        (setf nunit (char-equal diag #\N))
        (cond
         ((<= incx 0)
          (setf kx
                (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
         ((/= incx 1)
          (setf kx 1)))

```

```
(cond
((char-equal trans #\N)
(cond
(char-equal uplo #\U)
(setf kk 1)
(cond
(= incx 1)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(cond
(/= (f2cl-lib:fref x (j) ((1 *))) zero)
(setf temp
(f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(setf k kk)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i
(f2cl-lib:int-add j
(f2cl-lib:int-sub
1))))
nil)
(tagbody
(setf (f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
(+
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
(* temp
(f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%))))
(setf k (f2cl-lib:int-add k 1))))
(if nounit
(setf (f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)
(*
(f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)
(f2cl-lib:fref ap-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add kk j)
```

```

1))
((1 *))
ap-%offset%))))))
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%))
        (setf ix kx)
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
          (> k
            (f2cl-lib:int-add kk
              j
              (f2cl-lib:int-sub
                2))))
          nil)
        (tagbody
          (setf (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%))
            (+
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
              (* temp
                (f2cl-lib:fref ap-%data%
                  (k)
                  ((1 *))
                  ap-%offset%))))
            (setf ix (f2cl-lib:int-add ix incx))))
    (if nount
      (setf (f2cl-lib:fref x-%data%
        (jx)
        ((1 *))
        x-%offset%)
        (*
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)

```

```

(f2cl-lib:fref ap-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add kk j)
    1))
  ((1 *))
  ap-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf kk (the fixnum (truncate (* n (+ n 1) 2)))
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (setf k kk)
            (f2cl-lib:fdo (i n
              (f2cl-lib:int-add i
                (f2cl-lib:int-sub 1)))
              (> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
                x-%offset%)
                (+
                  (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%)
                  (* temp
                    (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%))))
                (setf k (f2cl-lib:int-sub k 1))))
            (if nunit
              (setf (f2cl-lib:fref x-%data%
                (j)
                ((1 *))
                x-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                    (j)
                    ((1 *))
                    x-%offset%)

```

```

(f2cl-lib:fref ap-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub kk n)
    j))
  ((1 *))
  ap-%offset%))))))
(setf kk
  (f2cl-lib:int-sub kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%))
          (setf ix kx)
          (f2cl-lib:fdo (k kk
            (f2cl-lib:int-add k
              (f2cl-lib:int-sub 1)))
            (> k
              (f2cl-lib:int-add kk
                (f2cl-lib:int-sub
                  (f2cl-lib:int-add
                    n
                      (f2cl-lib:int-sub
                        (f2cl-lib:int-add
                          j
                            1))))))
                nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%))
                (+
                  (f2cl-lib:fref x-%data%
                    (ix)

```

```

                                ((1 *))
                                x-%offset%)
    (* temp
      (f2cl-lib:fref ap-%data%
                    (k)
                    ((1 *))
                    ap-%offset%)))
    (setf ix (f2cl-lib:int-sub ix incx)))
  (if nunit
    (setf (f2cl-lib:fref x-%data%
                        (jx)
                        ((1 *))
                        x-%offset%)
      (*
        (f2cl-lib:fref x-%data%
                      (jx)
                      ((1 *))
                      x-%offset%)
        (f2cl-lib:fref ap-%data%
                      ((f2cl-lib:int-add
                        (f2cl-lib:int-sub kk n)
                        j))
                      ((1 *))
                      ap-%offset%))))))
    (setf jx (f2cl-lib:int-sub jx incx))
    (setf kk
      (f2cl-lib:int-sub kk
        (f2cl-lib:int-add
          (f2cl-lib:int-sub n j)
          1))))))
  (t
    (cond
      ((char-equal uplo #\U)
        (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                          ((> j 1) nil)
            (tagbody
              (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
              (if nunit
                (setf temp
                  (* temp
                    (f2cl-lib:fref ap-%data%
                                  (kk)
                                  ((1 *))
                                  ap-%offset%))))
              (setf k (f2cl-lib:int-sub kk 1))
              (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))

```

```

(f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
(> i 1) nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))
  (setf k (f2cl-lib:int-sub k 1)))
  (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
    temp)
  (setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix jx)
      (if nounit
        (setf temp
          (* temp
            (f2cl-lib:fref ap-%data%
              (kk)
              ((1 *))
              ap-%offset%))))
        (f2cl-lib:fdo (k
          (f2cl-lib:int-add kk (f2cl-lib:int-sub 1))
          (f2cl-lib:int-add k (f2cl-lib:int-sub 1))
          (> k
            (f2cl-lib:int-add kk
              (f2cl-lib:int-sub j)
              1))
          nil)
          (tagbody
            (setf ix (f2cl-lib:int-sub ix incx))
            (setf temp
              (+ temp
                (*

```



```

(f2cl-lib:fref ap-%data%
  (k)
  ((1 *))
  ap-%offset%)
(f2cl-lib:fref x-%data%
  (ix)
  ((1 *))
  x-%offset%))))))
(setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
  temp)
(setf jx (f2cl-lib:int-sub jx incx))
(setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf kk 1)
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (if nunit
          (setf temp
            (* temp
              (f2cl-lib:fref ap-%data%
                (kk)
                ((1 *))
                ap-%offset%))))
          (setf k (f2cl-lib:int-add kk 1))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
            (> i n) nil)
          (tagbody
            (setf temp
              (+ temp
                (*
                  (f2cl-lib:fref ap-%data%
                    (k)
                    ((1 *))
                    ap-%offset%)
                  (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%))))
              (setf k (f2cl-lib:int-add k 1))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
              temp)
          (setf kk
            (f2cl-lib:int-add kk
              (f2cl-lib:int-add

```

```

(f2c1-lib:int-sub n j)
1))))))

(t
  (setf jx kx)
  (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp
        (f2c1-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix jx)
      (if nunit
        (setf temp
          (* temp
            (f2c1-lib:fref ap-%data%
              (kk)
              ((1 *))
              ap-%offset%))))
        (f2c1-lib:fdo (k (f2c1-lib:int-add kk 1)
          (f2c1-lib:int-add k 1))
            ((> k
              (f2c1-lib:int-add kk
                n
                (f2c1-lib:int-sub j)))
              nil)
            (tagbody
              (setf ix (f2c1-lib:int-add ix incx))
              (setf temp
                (+ temp
                  (*
                    (f2c1-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
                    (f2c1-lib:fref x-%data%
                      (ix)
                      ((1 *))
                      x-%offset%))))))
              (setf (f2c1-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
                temp)
              (setf jx (f2c1-lib:int-add jx incx))
              (setf kk
                (f2c1-lib:int-add kk
                  (f2c1-lib:int-add
                    (f2c1-lib:int-sub n j)
                    1))))))))))

end_label
(return (values nil nil nil nil nil nil nil))))

```

dtpsv BLAS**— dtpsv.input —**

```

)set break resume
)sys rm -f dtpsv.output
)spool dtpsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtpsv.help —

```

=====
dtpsv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DTPSV - solve one of the systems of equations $Ax = b$, or $A'x = b$,

SYNOPSIS

SUBROUTINE DTPSV (UPLO, TRANS, DIAG, N, AP, X, INCX)

INTEGER INCX, N

CHARACTER*1 DIAG, TRANS, UPLO

DOUBLE PRECISION AP(*), X(*)

PURPOSE

DTPSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling

this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $A*x = b$.

TRANS = 'T' or 't' $A'*x = b$.

TRANS = 'C' or 'c' $A'*x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

AP - DOUBLE PRECISION array of DIMENSION at least $((n*(n+1))/2)$. Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains $a(1, 1)$, AP(2) and AP(3) contain $a(1, 2)$ and $a(2, 2)$ respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) con-

tains $a(1, 1)$, $AP(2)$ and $AP(3)$ contain $a(2, 1)$ and $a(3, 1)$ respectively, and so on. Note that when $DIAG = 'U'$ or $'u'$, the diagonal elements of A are not referenced, but are assumed to be unity. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element right-hand side vector b . On exit, X is overwritten with the solution vector x .

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X . INCX must not be zero. Unchanged on exit.

— dtpsv.f —

```

SUBROUTINE DTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )
*
* .. Scalar Arguments ..
  INTEGER          INCX, N
  CHARACTER*1      DIAG, TRANS, UPLO
*
* .. Array Arguments ..
  DOUBLE PRECISION AP( * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
  DOUBLE PRECISION ZERO
  PARAMETER        ( ZERO = 0.0D+0 )
*
* .. Local Scalars ..
  DOUBLE PRECISION TEMP
  INTEGER          I, INFO, IX, J, JX, K, KK, KX
  LOGICAL          NOUNIT
*
* .. External Functions ..
  LOGICAL          LSAME
  EXTERNAL         LSAME

```

```

*      .. External Subroutines ..
EXTERNAL          XERBLA
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$          .NOT.LSAME( UPLO , 'L' )      )THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$          .NOT.LSAME( TRANS, 'T' ).AND.
$          .NOT.LSAME( TRANS, 'C' )      )THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$          .NOT.LSAME( DIAG , 'N' )      )THEN
          INFO = 3
      ELSE IF( N.LT.0 )THEN
          INFO = 4
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 7
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'DTPSV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
      NOUNIT = LSAME( DIAG, 'N' )
*
*      Set up the start point in X if the increment is not unity. This
*      will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
          KX = 1
      END IF
*
*      Start the operations. In this version the elements of AP are
*      accessed sequentially with one pass through AP.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form  $x := inv(A)x$ .

```

```

*
      IF( LSAME( UPLO, 'U' ) )THEN
        KK = ( N*( N + 1 ) )/2
        IF( INCX.EQ.1 )THEN
          DO 20, J = N, 1, -1
            IF( X( J ).NE.ZERO )THEN
              IF( NUNIT )
$                X( J ) = X( J )/AP( KK )
              TEMP = X( J )
              K      = KK      - 1
              DO 10, I = J - 1, 1, -1
                X( I ) = X( I ) - TEMP*AP( K )
                K      = K      - 1
10              CONTINUE
            END IF
            KK = KK - J
20          CONTINUE
        ELSE
          JX = KK + ( N - 1 )*INCX
          DO 40, J = N, 1, -1
            IF( X( JX ).NE.ZERO )THEN
              IF( NUNIT )
$                X( JX ) = X( JX )/AP( KK )
              TEMP = X( JX )
              IX   = JX
              DO 30, K = KK - 1, KK - J + 1, -1
                IX   = IX   - INCX
                X( IX ) = X( IX ) - TEMP*AP( K )
30              CONTINUE
            END IF
            JX = JX - INCX
            KK = KK - J
40          CONTINUE
        END IF
      ELSE
        KK = 1
        IF( INCX.EQ.1 )THEN
          DO 60, J = 1, N
            IF( X( J ).NE.ZERO )THEN
              IF( NUNIT )
$                X( J ) = X( J )/AP( KK )
              TEMP = X( J )
              K      = KK      + 1
              DO 50, I = J + 1, N
                X( I ) = X( I ) - TEMP*AP( K )
                K      = K      + 1
50              CONTINUE
            END IF
            KK = KK + ( N - J + 1 )
60          CONTINUE

```

```

ELSE
  JX = KX
  DO 80, J = 1, N
    IF( X( JX ).NE.ZERO )THEN
      IF( NUNIT )
        $      X( JX ) = X( JX )/AP( KK )
      TEMP = X( JX )
      IX = JX
      DO 70, K = KK + 1, KK + N - J
        IX = IX + INCX
        X( IX ) = X( IX ) - TEMP*AP( K )
70      CONTINUE
      END IF
      JX = JX + INCX
      KK = KK + ( N - J + 1 )
80      CONTINUE
    END IF
  END IF
ELSE
*
*      Form x := inv( A' )*x.
*
  IF( LSAME( UPLO, 'U' ) )THEN
    KK = 1
    IF( INCX.EQ.1 )THEN
      DO 100, J = 1, N
        TEMP = X( J )
        K = KK
        DO 90, I = 1, J - 1
          TEMP = TEMP - AP( K )*X( I )
          K = K + 1
90      CONTINUE
        IF( NUNIT )
          $      TEMP = TEMP/AP( KK + J - 1 )
        X( J ) = TEMP
        KK = KK + J
100     CONTINUE
      ELSE
        JX = KX
        DO 120, J = 1, N
          TEMP = X( JX )
          IX = KX
          DO 110, K = KK, KK + J - 2
            TEMP = TEMP - AP( K )*X( IX )
            IX = IX + INCX
110     CONTINUE
          IF( NUNIT )
            $      TEMP = TEMP/AP( KK + J - 1 )
          X( JX ) = TEMP
          JX = JX + INCX

```



```

      KK      = KK  + J
120      CONTINUE
      END IF
      ELSE
      KK = ( N*( N + 1 ) )/2
      IF( INCX.EQ.1 )THEN
      DO 140, J = N, 1, -1
      TEMP = X( J )
      K = KK
      DO 130, I = N, J + 1, -1
      TEMP = TEMP - AP( K )*X( I )
      K = K - 1
130      CONTINUE
      IF( NUNIT )
      $      TEMP = TEMP/AP( KK - N + J )
      X( J ) = TEMP
      KK = KK - ( N - J + 1 )
140      CONTINUE
      ELSE
      KX = KX + ( N - 1 )*INCX
      JX = KX
      DO 160, J = N, 1, -1
      TEMP = X( JX )
      IX = KX
      DO 150, K = KK, KK - ( N - ( J + 1 ) ), -1
      TEMP = TEMP - AP( K )*X( IX )
      IX = IX - INCX
150      CONTINUE
      IF( NUNIT )
      $      TEMP = TEMP/AP( KK - N + J )
      X( JX ) = TEMP
      JX = JX - INCX
      KK = KK - ( N - J + 1 )
160      CONTINUE
      END IF
      END IF
      END IF
*
      RETURN
*
*      End of DTPSV .
*
      END

```

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtpsv (uplo trans diag n ap x incx)
    (declare (type (simple-array double-float (*)) x ap)
              (type fixnum incx n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (ap double-float ap-%data% ap-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nunit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0)
              (kx 0) (temp 0.0))
        (declare (type (member t nil) nunit)
                  (type fixnum i info ix j jx k kk kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          ((and (not (char-equal trans #\N))
                 (not (char-equal trans #\T))
                 (not (char-equal trans #\C)))
            (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
            (setf info 3))
          ((< n 0)
            (setf info 4))
          ((= incx 0)
            (setf info 7)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DTPSV" info)
            (go end_label)))
          (if (= n 0) (go end_label))
          (setf nunit (char-equal diag #\N))
          (cond
            ((<= incx 0)
              (setf kx
                (f2cl-lib:int-sub 1
                  (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                    incx))))
              ((/= incx 1)
                (setf kx 1)))
            (cond
              ((char-equal trans #\N)
                (cond

```

```

(char-equal uplo #\U)
(setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
(cond
  (= incx 1)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (if nounit
          (setf (f2cl-lib:fref x-%data%
                               (j)
                               ((1 *))
                               x-%offset%)
                (/
                 (f2cl-lib:fref x-%data%
                               (j)
                               ((1 *))
                               x-%offset%)
                 (f2cl-lib:fref ap-%data%
                               (kk)
                               ((1 *))
                               ap-%offset%))))))
      (setf temp
        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
      (setf k (f2cl-lib:int-sub kk 1))
      (f2cl-lib:fdo (i
                    (f2cl-lib:int-add j
                                       (f2cl-lib:int-sub 1))
                    (f2cl-lib:int-add i
                                       (f2cl-lib:int-sub 1)))
        (> i 1) nil)
      (tagbody
        (setf (f2cl-lib:fref x-%data%
                              (i)
                              ((1 *))
                              x-%offset%)
              (-
               (f2cl-lib:fref x-%data%
                              (i)
                              ((1 *))
                              x-%offset%)
               (* temp
                 (f2cl-lib:fref ap-%data%
                              (k)
                              ((1 *))
                              ap-%offset%))))))
        (setf k (f2cl-lib:int-sub k 1))))))
      (setf kk (f2cl-lib:int-sub kk j))))))
(t

```

```

(setf jx
  (f2cl-lib:int-add kx
    (f2cl-lib:int-mul
      (f2cl-lib:int-sub n 1)
      incx)))
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  ((> j 1) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (if nounit
          (setf (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%)
                (f2cl-lib:fref ap-%data%
                  (kk)
                  ((1 *))
                  ap-%offset%))))
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)))
        (setf ix jx)
        (f2cl-lib:fdo (k
          (f2cl-lib:int-add kk
            (f2cl-lib:int-sub 1))
          (f2cl-lib:int-add k
            (f2cl-lib:int-sub 1)))
          ((> k
            (f2cl-lib:int-add kk
              (f2cl-lib:int-sub
                j)
                1))
            nil)
          (tagbody
            (setf ix (f2cl-lib:int-sub ix incx))
            (setf (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%)
              (-
                (f2cl-lib:fref x-%data%
                  (ix)

```

```

                                ((1 *))
                                x-%offset%)
(* temp
  (f2cl-lib:fref ap-%data%
    (k)
    ((1 *))
    ap-%offset%))))))
  (setf jx (f2cl-lib:int-sub jx incx))
  (setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf kk 1)
  (cond
    ((= incx 1)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
       (> j n) nil)
     (tagbody
       (cond
         ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
          (if nounit
              (setf (f2cl-lib:fref x-%data%
                                (j)
                                ((1 *))
                                x-%offset%)
                    (/
                     (f2cl-lib:fref x-%data%
                                     (j)
                                     ((1 *))
                                     x-%offset%)
                     (f2cl-lib:fref ap-%data%
                                     (kk)
                                     ((1 *))
                                     ap-%offset%))))))
              (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
              (setf k (f2cl-lib:int-add kk 1))
              (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                            (f2cl-lib:int-add i 1))
                (> i n) nil)
              (tagbody
                (setf (f2cl-lib:fref x-%data%
                                    (i)
                                    ((1 *))
                                    x-%offset%)
                      (-
                       (f2cl-lib:fref x-%data%
                                       (i)
                                       ((1 *))
                                       x-%offset%)
                       (* temp
                        (f2cl-lib:fref ap-%data%

```

```

(k)
((1 *))
ap-%offset%)))
(setf k (f2cl-lib:int-add k 1))))))
(setf kk
  (f2cl-lib:int-add kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (if nunit
            (setf (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
                (f2cl-lib:fref ap-%data%
                                (kk)
                                ((1 *))
                                ap-%offset%))))))
          (setf temp
            (f2cl-lib:fref x-%data%
                            (jx)
                            ((1 *))
                            x-%offset%))
            (setf ix jx)
            (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                          (f2cl-lib:int-add k 1))
              ((> k
                (f2cl-lib:int-add kk
                  n
                  (f2cl-lib:int-sub
                    j)))
                nil)
              (tagbody
                (setf ix (f2cl-lib:int-add ix incx))
                (setf (f2cl-lib:fref x-%data%
                                    (ix)
                                    ((1 *))
                                    x-%offset%)

```

```

(-
  (f2cl-lib:fref x-%data%
                 (ix)
                 ((1 *))
                 x-%offset%)
  (* temp
    (f2cl-lib:fref ap-%data%
                   (k)
                   ((1 *))
                   ap-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk
  (f2cl-lib:int-add kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (cond
    ((char-equal uplo #\U)
      (setf kk 1)
      (cond
        ((= incx 1)
          (f2cl-lib:fdof (j 1 (f2cl-lib:int-add j 1))
            (> j n) nil)
          (tagbody
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (setf k kk)
            (f2cl-lib:fdof (i 1 (f2cl-lib:int-add i 1))
              (> i
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub 1)))
                nil)
            (tagbody
              (setf temp
                (- temp
                  (*
                    (f2cl-lib:fref ap-%data%
                                     (k)
                                     ((1 *))
                                     ap-%offset%)
                    (f2cl-lib:fref x-%data%
                                     (i)
                                     ((1 *))
                                     x-%offset%))))
                (setf k (f2cl-lib:int-add k 1))))
              (if nunit
                (setf temp
                  (/ temp
                    (f2cl-lib:fref ap-%data%

```

```

((f2cl-lib:int-sub
  (f2cl-lib:int-add kk j)
  1))
((1 *))
ap-%offset%))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
  temp)
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix kx)
    (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
      (> k
        (f2cl-lib:int-add kk
          j
          (f2cl-lib:int-sub 2)))
        nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:fref ap-%data%
              (k)
              ((1 *))
              ap-%offset%)
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))))
      (setf ix (f2cl-lib:int-add ix incx))))
  (if nount
    (setf temp
      (/ temp
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk j)
            1))
          ((1 *))
          ap-%offset%))))
    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
      temp)
    (setf jx (f2cl-lib:int-add jx incx))
    (setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf kk (the fixnum (truncate (* n (+ n 1) 2)))

```



```

(cond
  ((= incx 1)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
      (setf k kk)
      (f2cl-lib:fdo (i n
        (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
        (> i (f2cl-lib:int-add j 1)) nil)
        (tagbody
          (setf temp
            (- temp
              (*
                (f2cl-lib:fref ap-%data%
                  (k)
                  ((1 *))
                  ap-%offset%)
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%))))
            (setf k (f2cl-lib:int-sub k 1))))
          (if nunit
            (setf temp
              (/ temp
                (f2cl-lib:fref ap-%data%
                  ((f2cl-lib:int-add
                    (f2cl-lib:int-sub kk n)
                    j))
                  ((1 *))
                  ap-%offset%))))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
              temp)
            (setf kk
              (f2cl-lib:int-sub kk
                (f2cl-lib:int-add
                  (f2cl-lib:int-sub n j)
                  1))))))
    (t
      (setf kx
        (f2cl-lib:int-add kx
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub n 1)
            incx)))
      (setf jx kx)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j 1) nil)
      (tagbody

```

```

(setf temp
  (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
(setf ix kx)
(f2cl-lib:fdo (k kk
  (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
  (> k
    (f2cl-lib:int-add kk
      (f2cl-lib:int-sub
        (f2cl-lib:int-add n
          (f2cl-lib:int-sub
            (f2cl-lib:int-add
              j
              1))))))
    nil)
  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:fref ap-%data%
            (k)
            ((1 *))
            ap-%offset%)
          (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%))))))
    (setf ix (f2cl-lib:int-sub ix incx))))
(if nount
  (setf temp
    (/ temp
      (f2cl-lib:fref ap-%data%
        ((f2cl-lib:int-add
          (f2cl-lib:int-sub kk n)
          j))
        ((1 *))
        ap-%offset%))))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-sub jx incx))
  (setf kk
    (f2cl-lib:int-sub kk
      (f2cl-lib:int-add
        (f2cl-lib:int-sub n j)
        1))))))
end_label
(return (values nil nil nil nil nil nil nil))))

```

dtrmv BLAS**— dtrmv.input —**

```

)set break resume
)sys rm -f dtrmv.output
)spool dtrmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrmv.help —

```

=====
dtrmv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DTRMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$,

SYNOPSIS

SUBROUTINE DTRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

INTEGER INCX, LDA, N

CHARACTER*1 DIAG, TRANS, UPLO

DOUBLE PRECISION A(LDA, *), X(*)

PURPOSE

DTRMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an

upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := A*x$.

TRANS = 'T' or 't' $x := A'*x$.

TRANS = 'C' or 'c' $x := A'*x$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least $(1 + (n - 1) \cdot \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— dtrmv.f —

```

SUBROUTINE DTRMV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
*   .. Scalar Arguments ..
      INTEGER          INCX, LDA, N
      CHARACTER*1      DIAG, TRANS, UPLO
*   .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), X( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*      Jack Dongarra, Argonne National Lab.
*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.
*      Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
      DOUBLE PRECISION  ZERO
      PARAMETER          ( ZERO = 0.0D+0 )
*   .. Local Scalars ..
      DOUBLE PRECISION  TEMP
      INTEGER            I, INFO, IX, J, JX, KX
      LOGICAL            NOUNIT
*   .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*   .. External Subroutines ..
      EXTERNAL           XERBLA

```

```

*      .. Intrinsic Functions ..
      INTRINSIC          MAX
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$          .NOT.LSAME( UPLO , 'L' )          )THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$          .NOT.LSAME( TRANS, 'T' ).AND.
$          .NOT.LSAME( TRANS, 'C' )          )THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$          .NOT.LSAME( DIAG , 'N' )          )THEN
          INFO = 3
      ELSE IF( N.LT.0 )THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, N ) )THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 8
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'DTRMV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
      NOUNIT = LSAME( DIAG, 'N' )
*
*      Set up the start point in X if the increment is not unity. This
*      will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
          KX = 1
      END IF
*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through A.
*
      IF( LSAME( TRANS, 'N' ) )THEN

```

```

*
*      Form  x := A*x.
*
      IF( LSAME( UPLO, 'U' ) )THEN
        IF( INCX.EQ.1 )THEN
          DO 20, J = 1, N
            IF( X( J ).NE.ZERO )THEN
              TEMP = X( J )
              DO 10, I = 1, J - 1
                X( I ) = X( I ) + TEMP*A( I, J )
10             CONTINUE
              IF( NOUNIT )
                $      X( J ) = X( J )*A( J, J )
              END IF
            CONTINUE
20          ELSE
            JX = KX
            DO 40, J = 1, N
              IF( X( JX ).NE.ZERO )THEN
                TEMP = X( JX )
                IX = KX
                DO 30, I = 1, J - 1
                  X( IX ) = X( IX ) + TEMP*A( I, J )
                  IX = IX + INCX
30             CONTINUE
              IF( NOUNIT )
                $      X( JX ) = X( JX )*A( J, J )
              END IF
              JX = JX + INCX
            CONTINUE
40          END IF
        ELSE
          IF( INCX.EQ.1 )THEN
            DO 60, J = N, 1, -1
              IF( X( J ).NE.ZERO )THEN
                TEMP = X( J )
                DO 50, I = N, J + 1, -1
                  X( I ) = X( I ) + TEMP*A( I, J )
50             CONTINUE
              IF( NOUNIT )
                $      X( J ) = X( J )*A( J, J )
              END IF
            CONTINUE
60          ELSE
            KX = KX + ( N - 1 )*INCX
            JX = KX
            DO 80, J = N, 1, -1
              IF( X( JX ).NE.ZERO )THEN
                TEMP = X( JX )
                IX = KX

```

```

DO 70, I = N, J + 1, -1
    X( IX ) = X( IX ) + TEMP*A( I, J )
    IX      = IX      - INCX
70    CONTINUE
    IF( NUNIT )
$      X( JX ) = X( JX )*A( J, J )
    END IF
    JX = JX - INCX
80    CONTINUE
    END IF
    END IF
ELSE
*
*      Form x := A'*x.
*
    IF( LSAME( UPLO, 'U' ) )THEN
        IF( INCX.EQ.1 )THEN
            DO 100, J = N, 1, -1
                TEMP = X( J )
                IF( NUNIT )
$                    TEMP = TEMP*A( J, J )
                DO 90, I = J - 1, 1, -1
                    TEMP = TEMP + A( I, J )*X( I )
                CONTINUE
90                X( J ) = TEMP
            CONTINUE
100           CONTINUE
        ELSE
            JX = KX + ( N - 1 )*INCX
            DO 120, J = N, 1, -1
                TEMP = X( JX )
                IX    = JX
                IF( NUNIT )
$                    TEMP = TEMP*A( J, J )
                DO 110, I = J - 1, 1, -1
                    IX    = IX    - INCX
                    TEMP = TEMP + A( I, J )*X( IX )
                CONTINUE
110                X( JX ) = TEMP
                JX        = JX    - INCX
            CONTINUE
120           CONTINUE
        END IF
    ELSE
        IF( INCX.EQ.1 )THEN
            DO 140, J = 1, N
                TEMP = X( J )
                IF( NUNIT )
$                    TEMP = TEMP*A( J, J )
                DO 130, I = J + 1, N
                    TEMP = TEMP + A( I, J )*X( I )
                CONTINUE
130

```



```

          X( J ) = TEMP
140      CONTINUE
      ELSE
          JX = KX
          DO 160, J = 1, N
              TEMP = X( JX )
              IX = JX
              IF( NOUNIT )
$                  TEMP = TEMP*A( J, J )
              DO 150, I = J + 1, N
                  IX = IX + INCX
                  TEMP = TEMP + A( I, J )*X( IX )
150          CONTINUE
              X( JX ) = TEMP
              JX = JX + INCX
160      CONTINUE
          END IF
      END IF
  END IF
*
  RETURN
*
*   End of DTRMV .
*
  END

```

— BLAS 2 dtrmv —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrmv (uplo trans diag n a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
              (type fixnum incx lda n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nunit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type (member t nil) nunit)
                  (type fixnum i info ix j jx kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond

```

```
((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
      (setf info 1))
((and (not (char-equal trans #\N))
      (not (char-equal trans #\T))
      (not (char-equal trans #\C)))
      (setf info 2))
((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
      (setf info 3))
((< n 0)
      (setf info 4))
((< lda (max (the fixnum 1) (the fixnum n)))
      (setf info 6))
((= incx 0)
      (setf info 8)))
(cond
  ((/= info 0)
    (error
     " ** On entry to ~a parameter number ~a had an illegal value~%"
     "DTRMV" info)
    (go end_label)))
(if (= n 0) (go end_label))
(setf nounit (char-equal diag #\N))
(cond
  ((<= incx 0)
    (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                  incx))))
    ((/= incx 1)
      (setf kx 1)))
(cond
  ((char-equal trans #\N)
    (cond
      ((char-equal uplo #\U)
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          (> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                  (setf temp
                       (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                (> i
                                  (f2cl-lib:int-add j
                                                         (f2cl-lib:int-sub
                                                            1)))
                    nil)
                (tagbody
```

```

        (setf (f2cl-lib:fref x-%data%
                           (i)
                           ((1 *))
                           x-%offset%)
              (+
               (f2cl-lib:fref x-%data%
                              (i)
                              ((1 *))
                              x-%offset%)
               (* temp
                (f2cl-lib:fref a-%data%
                               (i j)
                               ((1 lda) (1 *))
                               a-%offset%))))))
(if nunit
    (setf (f2cl-lib:fref x-%data%
                       (j)
                       ((1 *))
                       x-%offset%)
          (*
           (f2cl-lib:fref x-%data%
                          (j)
                          ((1 *))
                          x-%offset%)
           (f2cl-lib:fref a-%data%
                          (j j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
(t
 (setf jx kx)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
 (tagbody
  (cond
   ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
    (setf temp
            (f2cl-lib:fref x-%data%
                           (jx)
                           ((1 *))
                           x-%offset%))
    (setf ix kx)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i
                     (f2cl-lib:int-add j
                               (f2cl-lib:int-sub
                                1)))
                   nil)
    (tagbody
     (setf (f2cl-lib:fref x-%data%
                          (ix)

```

```

((1 *))
x-%offset%)
(
  (f2cl-lib:fref x-%data%
    (ix)
    ((1 *))
    x-%offset%)
  (* temp
    (f2cl-lib:fref a-%data%
      (i j)
      ((1 lda) (1 *))
      a-%offset%)))
  (setf ix (f2cl-lib:int-add ix incx)))
(if nounit
  (setf (f2cl-lib:fref x-%data%
    (jx)
    ((1 *))
    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
        (jx)
        ((1 *))
        x-%offset%)
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%))))))
  (setf jx (f2cl-lib:int-add jx incx))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        ((> j 1) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
              (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
              (f2cl-lib:fdo (i n
                (f2cl-lib:int-add i
                  (f2cl-lib:int-sub 1)))
                ((> i (f2cl-lib:int-add j 1)) nil)
                (tagbody
                  (setf (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%)
                    (+
                      (f2cl-lib:fref x-%data%
                        (i)

```

```

                                ((1 *))
                                x-%offset%)
(* temp
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%))))))
(if nunit
  (setf (f2cl-lib:fref x-%data%
    (j)
    ((1 *))
    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
        (j)
        ((1 *))
        x-%offset%)
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%)))))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j 1) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%))
          (setf ix kx)
          (f2cl-lib:fdo (i n
            (f2cl-lib:int-add i
              (f2cl-lib:int-sub 1)))
            ((> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
                (+
                  (f2cl-lib:fref x-%data%

```

```

(ix)
((1 *))
x-%offset%)

(* temp
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)))
(setf ix (f2cl-lib:int-sub ix incx)))
(if nounit
  (setf (f2cl-lib:fref x-%data%
    (jx)
    ((1 *))
    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
        (jx)
        ((1 *))
        x-%offset%)
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%))))))
(setf jx (f2cl-lib:int-sub jx incx))))))

(t
  (cond
    ((char-equal uplo #\U)
      (cond
        ((= incx 1)
          (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
            (> j 1) nil)
          (tagbody
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (if nounit
              (setf temp
                (* temp
                  (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
              (> i 1) nil)
            (tagbody
              (setf temp
                (+ temp
                  (*
                    (f2cl-lib:fref a-%data%
                      (i j)

```

```

                                ((1 lda) (1 *))
                                a-%offset%)
                                (f2cl-lib:fref x-%data%
                                (i)
                                ((1 *))
                                x-%offset%))))))
                                (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
                                temp))))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix jx)
      (if nunit
        (setf temp
          (* temp
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%))))
        (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
          (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
          (> i 1) nil)
        (tagbody
          (setf ix (f2cl-lib:int-sub ix incx))
          (setf temp
            (+ temp
              (*
                (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref x-%data%
                  (ix)
                  ((1 *))
                  x-%offset%))))))
          (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
            temp)
          (setf jx (f2cl-lib:int-sub jx incx))))))
  (t
    (cond
      ((= incx 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```
((> j n) nil)
(tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
      (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%
        temp))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix jx)
    (if nounit
      (setf temp
        (* temp
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add i 1))
        (> i n) nil)
      (tagbody
        (setf ix (f2cl-lib:int-add ix incx))
        (setf temp
          (+ temp
```



```

      (*
      (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *)))
      a-%offset%)
      (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *)))
      x-%offset%))))))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
            temp)
      (setf jx (f2cl-lib:int-add jx incx)))))))))
end_label
      (return (values nil nil nil nil nil nil nil nil))))))

```

dtrsv BLAS

— dtrsv.input —

```

)set break resume
)sys rm -f dtrsv.output
)spool dtrsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrsv.help —

```

=====
dtrsv examples
=====

=====
Man Page Details
=====

NAME
    DTRSV - solve one of the systems of equations    A*x = b, or

```

$A'x = b$,

SYNOPSIS

SUBROUTINE DTRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

INTEGER INCX, LDA, N

CHARACTER*1 DIAG, TRANS, UPLO

DOUBLE PRECISION A(LDA, *), X(*)

PURPOSE

DTRSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $Ax = b$.

TRANS = 'T' or 't' $A'x = b$.

TRANS = 'C' or 'c' $A'x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit

triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X - DOUBLE PRECISION array of dimension at least (1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— dtrsv.f —

```

SUBROUTINE DTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
* .. Scalar Arguments ..
  INTEGER          INCX, LDA, N
  CHARACTER*1      DIAG, TRANS, UPLO

```

```

*      .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), X( * )
*
*      ..
*
*      Level 2 Blas routine.
*
*      -- Written on 22-October-1986.
*      Jack Dongarra, Argonne National Lab.
*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.
*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
      DOUBLE PRECISION  ZERO
      PARAMETER          ( ZERO = 0.0D+0 )
*
*      .. Local Scalars ..
      DOUBLE PRECISION  TEMP
      INTEGER           I, INFO, IX, J, JX, KX
      LOGICAL           NOUNIT
*
*      .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL          LSAME
*
*      .. External Subroutines ..
      EXTERNAL          XERBLA
*
*      .. Intrinsic Functions ..
      INTRINSIC         MAX
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$           .NOT.LSAME( UPLO , 'L' ) ) THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$           .NOT.LSAME( TRANS, 'T' ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$           .NOT.LSAME( DIAG , 'N' ) ) THEN
          INFO = 3
      ELSE IF( N.LT.0 ) THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 8
      END IF

```

```

      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'DTRSV ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( N.EQ.0 )
$    RETURN
*
      NOUNIT = LSAME( DIAG, 'N' )
*
*   Set up the start point in X if the increment is not unity. This
*   will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
        KX = 1
      END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through A.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*       Form x := inv( A )*x.
*
        IF( LSAME( UPLO, 'U' ) )THEN
          IF( INCX.EQ.1 )THEN
            DO 20, J = N, 1, -1
              IF( X( J ).NE.ZERO )THEN
                IF( NOUNIT )
$                  X( J ) = X( J )/A( J, J )
                TEMP = X( J )
                DO 10, I = J - 1, 1, -1
                  X( I ) = X( I ) - TEMP*A( I, J )
10              CONTINUE
              END IF
20            CONTINUE
          ELSE
            JX = KX + ( N - 1 )*INCX
            DO 40, J = N, 1, -1
              IF( X( JX ).NE.ZERO )THEN
                IF( NOUNIT )
$                  X( JX ) = X( JX )/A( J, J )
                TEMP = X( JX )
                IX = JX
                DO 30, I = J - 1, 1, -1
                  IX      = IX      - INCX

```

```

X( IX ) = X( IX ) - TEMP*A( I, J )
30      CONTINUE
      END IF
      JX = JX - INCX
40      CONTINUE
      END IF
    ELSE
      IF( INCX.EQ.1 )THEN
        DO 60, J = 1, N
          IF( X( J ).NE.ZERO )THEN
            IF( NOUNIT )
              $      X( J ) = X( J )/A( J, J )
              TEMP = X( J )
              DO 50, I = J + 1, N
                X( I ) = X( I ) - TEMP*A( I, J )
50              CONTINUE
              END IF
            CONTINUE
          ELSE
            JX = KX
            DO 80, J = 1, N
              IF( X( JX ).NE.ZERO )THEN
                IF( NOUNIT )
                  $      X( JX ) = X( JX )/A( J, J )
                  TEMP = X( JX )
                  IX = JX
                  DO 70, I = J + 1, N
                    IX = IX + INCX
                    X( IX ) = X( IX ) - TEMP*A( I, J )
70              CONTINUE
              END IF
              JX = JX + INCX
            CONTINUE
          END IF
        END IF
      ELSE
*
*      Form x := inv( A' ) * x.
*
      IF( LSAME( UPLO, 'U' ) )THEN
        IF( INCX.EQ.1 )THEN
          DO 100, J = 1, N
            TEMP = X( J )
            DO 90, I = 1, J - 1
              TEMP = TEMP - A( I, J ) * X( I )
90          CONTINUE
            IF( NOUNIT )
              $      TEMP = TEMP/A( J, J )
              X( J ) = TEMP
100         CONTINUE

```

```

        ELSE
            JX = KX
            DO 120, J = 1, N
                TEMP = X( JX )
                IX = KX
                DO 110, I = 1, J - 1
                    TEMP = TEMP - A( I, J ) * X( IX )
                    IX = IX + INCX
110                CONTINUE
                IF( NOUNIT )
                    $          TEMP = TEMP/A( J, J )
                    X( JX ) = TEMP
                    JX = JX + INCX
120                CONTINUE
            END IF
        ELSE
            IF( INCX.EQ.1 ) THEN
                DO 140, J = N, 1, -1
                    TEMP = X( J )
                    DO 130, I = N, J + 1, -1
                        TEMP = TEMP - A( I, J ) * X( I )
130                    CONTINUE
                    IF( NOUNIT )
                        $          TEMP = TEMP/A( J, J )
                        X( J ) = TEMP
140                CONTINUE
            ELSE
                KX = KX + ( N - 1 ) * INCX
                JX = KX
                DO 160, J = N, 1, -1
                    TEMP = X( JX )
                    IX = KX
                    DO 150, I = N, J + 1, -1
                        TEMP = TEMP - A( I, J ) * X( IX )
                        IX = IX - INCX
150                    CONTINUE
                    IF( NOUNIT )
                        $          TEMP = TEMP/A( J, J )
                        X( JX ) = TEMP
                        JX = JX - INCX
160                CONTINUE
            END IF
        END IF
    END IF
*
    RETURN
*
*   End of DTRSV .
*
END

```

— BLAS 2 dtrsv —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrsv (uplo trans diag n a lda x incx)
    (declare (type (simple-array double-float (*)) x a)
              (type fixnum incx lda n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (x double-float x-%data% x-%offset%))
      (prog ((nunit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp 0.0))
        (declare (type (member t nil) nunit)
                  (type fixnum i info ix j jx kx)
                  (type (double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
          (setf info 2))
         ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
          (setf info 3))
         ((< n 0)
          (setf info 4))
         ((< lda (max (the fixnum 1) (the fixnum n)))
          (setf info 6))
         ((= incx 0)
          (setf info 8)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "DTRSV" info)
          (go end_label)))
        (if (= n 0) (go end_label))
        (setf nunit (char-equal diag #\N))
        (cond
         ((<= incx 0)
          (setf kx

```



```

(f2cl-lib:int-sub 1
  (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
    incx)))

( (/= incx 1)
  (setf kx 1)))
(cond
  ((char-equal trans #\N)
    (cond
      ((char-equal uplo #\U)
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j 1) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                  (if nounit
                    (setf (f2cl-lib:fref x-%data%
                      (j)
                      ((1 *))
                      x-%offset%)
                      (/
                        (f2cl-lib:fref x-%data%
                          (j)
                          ((1 *))
                          x-%offset%)
                        (f2cl-lib:fref a-%data%
                          (j j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
                  (setf temp
                    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (f2cl-lib:fdo (i
                    (f2cl-lib:int-add j
                      (f2cl-lib:int-sub 1))
                    (f2cl-lib:int-add i
                      (f2cl-lib:int-sub 1)))
                      ((> i 1) nil)
                    (tagbody
                      (setf (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%)
                        (-
                          (f2cl-lib:fref x-%data%
                            (i)
                            ((1 *))
                            x-%offset%)
                          (* temp
                            (f2cl-lib:fref a-%data%

```

```

(i j)
((1 lda) (1 *))
a-%offset%))))))))))

(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (if nounit
            (setf (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%)
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%))))))
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)))
          (setf ix jx)
          (f2cl-lib:fdo (i
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub 1))
            (f2cl-lib:int-add i
              (f2cl-lib:int-sub 1)))
            (> i 1) nil)
            (tagbody
              (setf ix (f2cl-lib:int-sub ix incx))
              (setf (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
                (-
                  (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))

```

```
(t
(cond
  ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
          (if nounit
            (setf (f2cl-lib:fref x-%data%
              (j)
              ((1 *))
              x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                  (j)
                  ((1 *))
                  x-%offset%)
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%))))))
          (setf temp
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
            (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%)
              (-
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%)
                (* temp
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%))))))))))
(t
```

```

(setf jx kx)
(f2cl-lib:fdof (j 1 (f2cl-lib:int-add j 1))
  ((> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (if nounit
          (setf (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
                (/
                  (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
                  (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%))))))
      (setf temp
        (f2cl-lib:fref x-%data%
                        (jx)
                        ((1 *))
                        x-%offset%))
      (setf ix jx)
      (f2cl-lib:fdof (i (f2cl-lib:int-add j 1)
                        (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf ix (f2cl-lib:int-add ix incx))
          (setf (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                (-
                  (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                  (* temp
                    (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%))))))
          (setf jx (f2cl-lib:int-add jx incx))))))
    (t
      (cond
        ((char-equal uplo #\U)
          (cond

```

```

(= incx 1)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub 1)))
    nil)
  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:fref a-%data%
            (i j)
            ((1 lda) (1 *))
            a-%offset%)
          (f2cl-lib:fref x-%data%
            (i)
            ((1 *))
            x-%offset%))))))
  (if nount
    (setf temp
      (/ temp
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp)))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix kx)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
    (if nount
      (setf temp
        (/ temp
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%)))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
        temp)))
  )

```

```

                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                                (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%))))
                                (setf ix (f2cl-lib:int-add ix incx)))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%)))
    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
      temp)
    (setf jx (f2cl-lib:int-add jx incx))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        ((> j 1) nil)
        (tagbody
          (setf temp
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
          (f2cl-lib:fdo (i n
            (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
              ((> i (f2cl-lib:int-add j 1)) nil)
              (tagbody
                (setf temp
                  (- temp
                    (*
                      (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)
                      (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%)))))))
          (if nunit
            (setf temp
              (/ temp
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%)))
            (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
              temp))))))

```

```

(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix kx)
      (f2cl-lib:fdo (i n
        (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
        ((> i (f2cl-lib:int-add j 1)) nil)
        (tagbody
          (setf temp
            (- temp
              (*
                (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref x-%data%
                  (ix)
                  ((1 *))
                  x-%offset%))))
            (setf ix (f2cl-lib:int-sub ix incx))))
          (if nunit
            (setf temp
              (/ temp
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%))))
            (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
              temp)
            (setf jx (f2cl-lib:int-sub jx incx))))))))
  end_label
  (return (values nil nil nil nil nil nil nil nil))))

```

zgbmv BLAS

— zgbmv.input —

```

)set break resume
)sys rm -f zgbmv.output
)spool zgbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zgbmv.help —

```

=====
zgbmv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZGBMV - perform one of the matrix-vector operations $y := \alpha A x + \beta y$, or $y := \alpha A' x + \beta y$, or $y := \alpha \text{conjg}(A') x + \beta y$,

SYNOPSIS

```

SUBROUTINE ZGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X,
                  INCX, BETA, Y, INCY )

```

```

      COMPLEX*16  ALPHA, BETA

```

```

      INTEGER     INCX, INCY, KL, KU, LDA, M, N

```

```

      CHARACTER*1 TRANS

```

```

      COMPLEX*16  A( LDA, * ), X( * ), Y( * )

```

PURPOSE

ZGBMV performs one of the matrix-vector operations

where alpha and beta are scalars, x and y are vectors and A is an m by n band matrix, with kl sub-diagonals and ku super-diagonals.

PARAMETERS

TRANS - CHARACTER*1.
 On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $y := \alpha * A * x + \beta * y.$

TRANS = 'T' or 't' $y := \alpha * A' * x + \beta * y.$

TRANS = 'C' or 'c' $y := \alpha * \text{conjg}(A') * x + \beta * y.$

Unchanged on exit.

M - INTEGER.
 On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.
 On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

KL - INTEGER.
 On entry, KL specifies the number of sub-diagonals of the matrix A. KL must satisfy $0 \leq KL$. Unchanged on exit.

KU - INTEGER.
 On entry, KU specifies the number of super-diagonals of the matrix A. KU must satisfy $0 \leq KU$. Unchanged on exit.

ALPHA - COMPLEX*16
 On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
 Before entry, the leading (kl + ku + 1) by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (ku + 1) of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row (ku + 2), and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced. The following program segment will transfer a band matrix from conventional full

matrix storage to band storage:

```
DO 20, J = 1, N K = KU + 1 - J DO 10, I = MAX( 1, J -
KU ), MIN( M, J + KL ) A( K + I, J ) = matrix( I, J )
10    CONTINUE 20 CONTINUE
```

Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (kl + ku + 1). Unchanged on exit.

X - COMPLEX*16 array of DIMENSION at least
(1 + (n - 1) * abs(INCX)) when TRANS = 'N' or 'n'
and at least (1 + (m - 1) * abs(INCX)) otherwise.
Before entry, the incremented array X must contain
the vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - COMPLEX*16 .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y - COMPLEX*16 array of DIMENSION at least
(1 + (m - 1) * abs(INCY)) when TRANS = 'N' or 'n'
and at least (1 + (n - 1) * abs(INCY)) otherwise.
Before entry, the incremented array Y must contain
the vector y. On exit, Y is overwritten by the
updated vector y.

INCY - INTEGER.

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

— zgbmv.f —

```
SUBROUTINE ZGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
```

* .. Scalar Arguments ..

```

      COMPLEX*16      ALPHA, BETA
      INTEGER         INCX, INCY, KL, KU, LDA, M, N
      CHARACTER*1     TRANS
*   .. Array Arguments ..
      COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*   .. Parameters ..
      COMPLEX*16      ONE
      PARAMETER       ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16      ZERO
      PARAMETER       ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*   .. Local Scalars ..
      COMPLEX*16      TEMP
      INTEGER         I, INFO, IX, IY, J, JX, JY, K, KUP1, KX, KY,
$      LENX, LENY
      LOGICAL         NOCONJ
*   .. External Functions ..
      LOGICAL         LSAME
      EXTERNAL        LSAME
*   .. External Subroutines ..
      EXTERNAL        XERBLA
*   .. Intrinsic Functions ..
      INTRINSIC       DCONJG, MAX, MIN
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( TRANS, 'N' ) ).AND.
$           .NOT.LSAME( TRANS, 'T' ) ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 1
      ELSE IF( M.LT.0 ) THEN
          INFO = 2
      ELSE IF( N.LT.0 ) THEN
          INFO = 3
      ELSE IF( KL.LT.0 ) THEN
          INFO = 4
      ELSE IF( KU.LT.0 ) THEN

```

```

        INFO = 5
      ELSE IF( LDA.LT.( KL + KU + 1 ) )THEN
        INFO = 8
      ELSE IF( INCX.EQ.0 )THEN
        INFO = 10
      ELSE IF( INCY.EQ.0 )THEN
        INFO = 13
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZGBMV ', INFO )
        RETURN
      END IF

*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$      ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
      NOCONJ = LSAME( TRANS, 'T' )
*
*   Set LENX and LENY, the lengths of the vectors x and y, and set
*   up the start points in X and Y.
*
      IF( LSAME( TRANS, 'N' ) )THEN
        LENX = N
        LENY = M
      ELSE
        LENX = M
        LENY = N
      END IF
      IF( INCX.GT.0 )THEN
        KX = 1
      ELSE
        KX = 1 - ( LENX - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( LENY - 1 )*INCY
      END IF

*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through the band part of A.
*
*   First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN

```

```

        DO 10, I = 1, LENY
            Y( I ) = ZERO
10        CONTINUE
        ELSE
            DO 20, I = 1, LENY
                Y( I ) = BETA*Y( I )
20        CONTINUE
            END IF
        ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
                DO 30, I = 1, LENY
                    Y( IY ) = ZERO
                    IY      = IY  + INCY
30        CONTINUE
                ELSE
                    DO 40, I = 1, LENY
                        Y( IY ) = BETA*Y( IY )
                        IY      = IY  + INCY
40        CONTINUE
                    END IF
                END IF
            END IF
            IF( ALPHA.EQ.ZERO )
$      RETURN
            KUP1 = KU + 1
            IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form y := alpha*A*x + y.
*
            JX = KX
            IF( INCY.EQ.1 )THEN
                DO 60, J = 1, N
                    IF( X( JX ).NE.ZERO )THEN
                        TEMP = ALPHA*X( JX )
                        K      = KUP1 - J
                        DO 50, I = MAX( 1, J - KU ), MIN( M, J + KL )
                            Y( I ) = Y( I ) + TEMP*A( K + I, J )
50        CONTINUE
                        END IF
                        JX = JX + INCX
60        CONTINUE
                    ELSE
                        DO 80, J = 1, N
                            IF( X( JX ).NE.ZERO )THEN
                                TEMP = ALPHA*X( JX )
                                IY      = KY
                                K      = KUP1 - J
                                DO 70, I = MAX( 1, J - KU ), MIN( M, J + KL )
                                    Y( IY ) = Y( IY ) + TEMP*A( K + I, J )

```

```

              IY      = IY      + INCY
70          CONTINUE
          END IF
          JX = JX + INCX
          IF( J.GT.KU )
$           KY = KY + INCY
80      CONTINUE
      END IF
  ELSE
*
*      Form y := alpha*A'*x + y or y := alpha*conjg( A' )*x + y.
*
      JY = KY
      IF( INCX.EQ.1 )THEN
          DO 110, J = 1, N
              TEMP = ZERO
              K      = KUP1 - J
              IF( NOCONJ )THEN
                  DO 90, I = MAX( 1, J - KU ), MIN( M, J + KL )
                      TEMP = TEMP + A( K + I, J ) * X( I )
90              CONTINUE
              ELSE
                  DO 100, I = MAX( 1, J - KU ), MIN( M, J + KL )
                      TEMP = TEMP + DCONJG( A( K + I, J ) ) * X( I )
100             CONTINUE
              END IF
              Y( JY ) = Y( JY ) + ALPHA * TEMP
              JY      = JY      + INCY
110          CONTINUE
      ELSE
          DO 140, J = 1, N
              TEMP = ZERO
              IX      = KX
              K      = KUP1 - J
              IF( NOCONJ )THEN
                  DO 120, I = MAX( 1, J - KU ), MIN( M, J + KL )
                      TEMP = TEMP + A( K + I, J ) * X( IX )
                      IX      = IX      + INCX
120             CONTINUE
              ELSE
                  DO 130, I = MAX( 1, J - KU ), MIN( M, J + KL )
                      TEMP = TEMP + DCONJG( A( K + I, J ) ) * X( IX )
                      IX      = IX      + INCX
130             CONTINUE
              END IF
              Y( JY ) = Y( JY ) + ALPHA * TEMP
              JY      = JY      + INCY
              IF( J.GT.KU )
$               KX = KX + INCX
140          CONTINUE

```

```

        END IF
    END IF
*
    RETURN
*
*   End of ZGBMV .
*
END

```

— BLAS 2 zgbmv —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zgbmv (trans m n kl ku alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
      (type (complex double-float) beta alpha)
      (type fixnum incy incx lda ku kl n m)
      (type character trans))
    (f2cl-lib:with-multi-array-data
      ((trans character trans-%data% trans-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%))
      (prog ((noconj nil) (i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0)
             (k 0) (kup1 0) (kx 0) (ky 0) (lenx 0) (leny 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj)
          (type fixnum i info ix iy j jx jy k kup1 kx ky
                    lenx leny)
          (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 1))
          ((< m 0)
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< kl 0)
           (setf info 4))
          ((< ku 0)
           (setf info 5))
          ((< lda (f2cl-lib:int-add kl ku 1))
           (setf info 8))
          ((= incx 0)

```

```

      (setf info 10))
      ((= incy 0)
       (setf info 13)))
    (cond
      ((/= info 0)
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "ZGBMV" info)
       (go end_label)))
    (if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
        (go end_label))
    (setf noconj (char-equal trans #\T))
    (cond
      ((char-equal trans #\N)
       (setf lenx n)
       (setf leny m)
       (t
        (setf lenx m)
        (setf leny n)))
      (cond
        ((> incx 0)
         (setf kx 1))
        (t
         (setf kx
          (f2cl-lib:int-sub 1
           (f2cl-lib:int-mul
            (f2cl-lib:int-sub lenx 1)
            incx))))))
      (cond
        ((> incy 0)
         (setf ky 1))
        (t
         (setf ky
          (f2cl-lib:int-sub 1
           (f2cl-lib:int-mul
            (f2cl-lib:int-sub leny 1)
            incy))))))
      (cond
        ((/= beta one)
         (cond
          ((= incy 1)
           (cond
            ((= beta zero)
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                           ((> i leny) nil)
              (tagbody
               (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                     zero))))
            (t
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))

```



```

                                (> i leny) nil)
      (tagbody
        (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
          (* beta
            (f2cl-lib:fref y-%data%
                          (i)
                          ((1 *))
                          y-%offset%))))))
    (t
      (setf iy ky)
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i leny) nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
              zero)
            (setf iy (f2cl-lib:int-add iy incy))))))
      (t
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i leny) nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
            (* beta
              (f2cl-lib:fref y-%data%
                            (iy)
                            ((1 *))
                            y-%offset%)))
          (setf iy (f2cl-lib:int-add iy incy))))))
    (if (= alpha zero) (go end_label))
    (setf kup1 (f2cl-lib:int-add ku 1))
    (cond
      ((char-equal trans #\N)
        (setf jx kx)
        (cond
          ((= incy 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
                  (setf temp
                    (* alpha
                      (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)))
                  (setf k (f2cl-lib:int-sub kup1 j))
                  (f2cl-lib:fdo (i
                                (max (the fixnum 1)

```

```

                                (the fixnum
                                (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                ku))))
                                (f2cl-lib:int-add i 1))
                                (> i
                                (min (the fixnum m)
                                (the fixnum
                                (f2cl-lib:int-add j kl))))
                                nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data%
        (i)
        ((1 *))
        y-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add k i) j)
          ((1 lda) (1 *))
          a-%offset%))))))
  (setf jx (f2cl-lib:int-add jx incx))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)))
        (setf iy ky)
        (setf k (f2cl-lib:int-sub kup1 j))
        (f2cl-lib:fdo (i
          (max (the fixnum 1)
            (the fixnum
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub
                ku))))
          (f2cl-lib:int-add i 1))
          (> i
          (min (the fixnum m)
            (the fixnum
              (f2cl-lib:int-add j kl))))
          nil)
        (tagbody

```

```

      (setf (f2c1-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (+
          (f2c1-lib:fref y-%data%
            (iy)
            ((1 *))
            y-%offset%)
          (* temp
            (f2c1-lib:fref a-%data%
              ((f2c1-lib:int-add k i) j)
              ((1 lda) (1 *))
              a-%offset%))))))
      (setf iy (f2c1-lib:int-add iy incy))))))
    (setf jx (f2c1-lib:int-add jx incx))
    (if (> j ku) (setf ky (f2c1-lib:int-add ky incy)))))))))
(t
  (setf jy ky)
  (cond
    ((= incx 1)
      (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf temp zero)
          (setf k (f2c1-lib:int-sub kup1 j))
          (cond
            (noconj
              (f2c1-lib:fdo (i
                (max (the fixnum 1)
                  (the fixnum
                    (f2c1-lib:int-add j
                      (f2c1-lib:int-sub
                        ku))))
                  (f2c1-lib:int-add i 1))
                ((> i
                  (min (the fixnum m)
                    (the fixnum
                      (f2c1-lib:int-add j kl))))
                  nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2c1-lib:fref a-%data%
                          ((f2c1-lib:int-add k i) j)
                          ((1 lda) (1 *))
                          a-%offset%)
                        (f2c1-lib:fref x-%data%
                          (i)
                          ((1 *))
                          x-%offset%))))))))))
    ))
  (t
    (setf jy ky)
    (cond
      ((= incx 1)
        (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
          ((> j n) nil)
          (tagbody
            (setf temp zero)
            (setf k (f2c1-lib:int-sub kup1 j))
            (cond
              (noconj
                (f2c1-lib:fdo (i
                  (max (the fixnum 1)
                    (the fixnum
                      (f2c1-lib:int-add j
                        (f2c1-lib:int-sub
                          ku))))
                    (f2c1-lib:int-add i 1))
                  ((> i
                    (min (the fixnum m)
                      (the fixnum
                        (f2c1-lib:int-add j kl))))
                    nil)
                  (tagbody
                    (setf temp
                      (+ temp
                        (*
                          (f2c1-lib:fref a-%data%
                            ((f2c1-lib:int-add k i) j)
                            ((1 lda) (1 *))
                            a-%offset%)
                          (f2c1-lib:fref x-%data%
                            (i)
                            ((1 *))
                            x-%offset%))))))))
              ))
            ))
        ))
      ((= incy 1)
        (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
          ((> i n) nil)
          (tagbody
            (setf temp zero)
            (setf ky (f2c1-lib:int-sub kupy1 i))
            (cond
              (noconj
                (f2c1-lib:fdo (j
                  (max (the fixnum 1)
                    (the fixnum
                      (f2c1-lib:int-add i
                        (f2c1-lib:int-sub
                          ky))))
                    (f2c1-lib:int-add j 1))
                  ((> j
                    (min (the fixnum m)
                      (the fixnum
                        (f2c1-lib:int-add i kl))))
                    nil)
                  (tagbody
                    (setf temp
                      (+ temp
                        (*
                          (f2c1-lib:fref a-%data%
                            ((f2c1-lib:int-add k i) j)
                            ((1 lda) (1 *))
                            a-%offset%)
                          (f2c1-lib:fref x-%data%
                            (j)
                            ((1 *))
                            x-%offset%))))))))
              ))
            ))
        ))
      ((= incx incy)
        (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
          ((> i n) nil)
          (tagbody
            (setf temp zero)
            (setf ky (f2c1-lib:int-sub kupy1 i))
            (cond
              (noconj
                (f2c1-lib:fdo (j
                  (max (the fixnum 1)
                    (the fixnum
                      (f2c1-lib:int-add i
                        (f2c1-lib:int-sub
                          ky))))
                    (f2c1-lib:int-add j 1))
                  ((> j
                    (min (the fixnum m)
                      (the fixnum
                        (f2c1-lib:int-add i kl))))
                    nil)
                  (tagbody
                    (setf temp
                      (+ temp
                        (*
                          (f2c1-lib:fref a-%data%
                            ((f2c1-lib:int-add k i) j)
                            ((1 lda) (1 *))
                            a-%offset%)
                          (f2c1-lib:fref x-%data%
                            (j)
                            ((1 *))
                            x-%offset%))))))))
              ))
            ))
        ))
      ))
    ))
  )
)

```

```

(f2cl-lib:fdo (i
  (max (the fixnum 1)
        (the fixnum
          (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                  ku))))
    (f2cl-lib:int-add i 1))
  (> i
    (min (the fixnum m)
          (the fixnum
            (f2cl-lib:int-add j kl))))
  nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add k i) j)
                        ((1 lda) (1 *))
                        a-%offset%))
        (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%))))))
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (* alpha temp)))
  (setf jy (f2cl-lib:int-add jy incy))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp zero)
    (setf ix kx)
    (setf k (f2cl-lib:int-sub kup1 j))
    (cond
      (noconj
        (f2cl-lib:fdo (i
          (max (the fixnum 1)
                (the fixnum
                  (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                  ku))))
            (f2cl-lib:int-add i 1))
          (> i
            (min (the fixnum m)
                  (the fixnum
                    (f2cl-lib:int-add j kl))))
          nil)

```

```

(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add k i) j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))
    (setf ix (f2cl-lib:int-add ix incx))))
(t
  (f2cl-lib:fdo (i
    (max (the fixnum 1)
      (the fixnum
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub
            ku))))
    (f2cl-lib:int-add i 1))
    (> i
      (min (the fixnum m)
        (the fixnum
          (f2cl-lib:int-add j kl))))
    nil)
  (tagbody
    (setf temp
      (+ temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              ((f2cl-lib:int-add k i) j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))))))
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (* alpha temp)))
  (setf jy (f2cl-lib:int-add jy incy))
  (if (> j ku) (setf kx (f2cl-lib:int-add kx incx))))))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil))))

```

zgemv BLAS

— zgemv.input —

```
)set break resume
)sys rm -f zgemv.output
)spool zgemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zgemv.help —

```
=====
zgemv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZGEMV - perform one of the matrix-vector operations $y := \alpha A x + \beta y$, or $y := \alpha A' x + \beta y$, or $y := \alpha \text{conj}(A') x + \beta y$,

SYNOPSIS

```
SUBROUTINE ZGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
                  BETA, Y, INCY )
```

```
COMPLEX*16  ALPHA, BETA
```

```
INTEGER     INCX, INCY, LDA, M, N
```

```
CHARACTER*1 TRANS
```

```
COMPLEX*16  A( LDA, * ), X( * ), Y( * )
```

PURPOSE

ZGEMV performs one of the matrix-vector operations

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

PARAMETERS

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $y := \alpha * A * x + \beta * y.$

TRANS = 'T' or 't' $y := \alpha * A' * x + \beta * y.$

TRANS = 'C' or 'c' $y := \alpha * \text{conjg}(A') * x + \beta * y.$

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).

Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, m)$. Unchanged on exit.

X - COMPLEX*16 array of DIMENSION at least
 $(1 + (n - 1) * \text{abs}(\text{INCX}))$ when TRANS = 'N' or 'n'
 and at least $(1 + (m - 1) * \text{abs}(\text{INCX}))$ otherwise.
 Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y - COMPLEX*16 array of DIMENSION at least
(1 + (m - 1) * abs(INCY)) when TRANS = 'N' or 'n'
and at least (1 + (n - 1) * abs(INCY)) otherwise.
Before entry with BETA non-zero, the incremented
array Y must contain the vector y. On exit, Y is
overwritten by the updated vector y.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

— zgemv.f —

```

SUBROUTINE ZGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
*   .. Scalar Arguments ..
COMPLEX*16      ALPHA, BETA
INTEGER         INCX, INCY, LDA, M, N
CHARACTER*1     TRANS
*   .. Array Arguments ..
COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
COMPLEX*16      ONE
PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16      ZERO

```



```

      PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*   .. Local Scalars ..
      COMPLEX*16      TEMP
      INTEGER         I, INFO, IX, IY, J, JX, JY, KX, KY, LENX, LENY
      LOGICAL         NOCONJ
*   .. External Functions ..
      LOGICAL         LSAME
      EXTERNAL        LSAME
*   .. External Subroutines ..
      EXTERNAL        XERBLA
*   .. Intrinsic Functions ..
      INTRINSIC       DCONJG, MAX
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( TRANS, 'N' ) ).AND.
$           .NOT.LSAME( TRANS, 'T' ) ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 1
      ELSE IF( M.LT.0 ) THEN
          INFO = 2
      ELSE IF( N.LT.0 ) THEN
          INFO = 3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 8
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 11
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZGEMV ', INFO )
          RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$       ( ( ALPHA.EQ.ZERO ) .AND.( BETA.EQ.ONE ) ) )
$       RETURN
*
      NOCONJ = LSAME( TRANS, 'T' )
*
*   Set LENX and LENY, the lengths of the vectors x and y, and set
*   up the start points in X and Y.
*
      IF( LSAME( TRANS, 'N' ) ) THEN

```

```

        LENX = N
        LENY = M
    ELSE
        LENX = M
        LENY = N
    END IF
    IF( INCX.GT.0 )THEN
        KX = 1
    ELSE
        KX = 1 - ( LENX - 1 )*INCX
    END IF
    IF( INCY.GT.0 )THEN
        KY = 1
    ELSE
        KY = 1 - ( LENY - 1 )*INCY
    END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through A.
*
*   First form  y := beta*y.
*
    IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
            IF( BETA.EQ.ZERO )THEN
                DO 10, I = 1, LENY
                    Y( I ) = ZERO
10             CONTINUE
            ELSE
                DO 20, I = 1, LENY
                    Y( I ) = BETA*Y( I )
20             CONTINUE
            END IF
        ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
                DO 30, I = 1, LENY
                    Y( IY ) = ZERO
                    IY      = IY  + INCY
30             CONTINUE
            ELSE
                DO 40, I = 1, LENY
                    Y( IY ) = BETA*Y( IY )
                    IY      = IY  + INCY
40             CONTINUE
            END IF
        END IF
    END IF
    IF( ALPHA.EQ.ZERO )
$   RETURN

```

```

      IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form y := alpha*A*x + y.
*
      JX = KX
      IF( INCY.EQ.1 )THEN
        DO 60, J = 1, N
          IF( X( JX ).NE.ZERO )THEN
            TEMP = ALPHA*X( JX )
            DO 50, I = 1, M
              Y( I ) = Y( I ) + TEMP*A( I, J )
50          CONTINUE
            END IF
            JX = JX + INCX
60        CONTINUE
      ELSE
        DO 80, J = 1, N
          IF( X( JX ).NE.ZERO )THEN
            TEMP = ALPHA*X( JX )
            IY = KY
            DO 70, I = 1, M
              Y( IY ) = Y( IY ) + TEMP*A( I, J )
              IY = IY + INCY
70          CONTINUE
            END IF
            JX = JX + INCX
80        CONTINUE
      END IF
      ELSE
*
*      Form y := alpha*A'*x + y or y := alpha*conjg( A' )*x + y.
*
      JY = KY
      IF( INCX.EQ.1 )THEN
        DO 110, J = 1, N
          TEMP = ZERO
          IF( NOCONJ )THEN
            DO 90, I = 1, M
              TEMP = TEMP + A( I, J )*X( I )
90          CONTINUE
          ELSE
            DO 100, I = 1, M
              TEMP = TEMP + DCONJG( A( I, J ) )*X( I )
100         CONTINUE
          END IF
          Y( JY ) = Y( JY ) + ALPHA*TEMP
          JY = JY + INCY
110        CONTINUE
      ELSE
        DO 140, J = 1, N

```

```

      TEMP = ZERO
      IX   = KX
      IF( NOCONJ )THEN
        DO 120, I = 1, M
          TEMP = TEMP + A( I, J ) * X( IX )
          IX   = IX   + INCX
120      CONTINUE
      ELSE
        DO 130, I = 1, M
          TEMP = TEMP + DCONJG( A( I, J ) ) * X( IX )
          IX   = IX   + INCX
130      CONTINUE
      END IF
      Y( JY ) = Y( JY ) + ALPHA * TEMP
      JY      = JY      + INCY
140      CONTINUE
      END IF
    END IF
*
    RETURN
*
*   End of ZGEMV .
*
    END

```

— BLAS 2 zgemv —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
(declare (type (complex double-float) one) (type (complex double-float) zero))
(defun zgemv (trans m n alpha a lda x incx beta y incy)
  (declare (type (simple-array (complex double-float) (*)) y x a)
    (type (complex double-float) beta alpha)
    (type fixnum incy incx lda n m)
    (type character trans))
  (f2cl-lib:with-multi-array-data
    ((trans character trans-%data% trans-%offset%)
     (a (complex double-float) a-%data% a-%offset%)
     (x (complex double-float) x-%data% x-%offset%)
     (y (complex double-float) y-%data% y-%offset%))
    (prog ((noconj nil) (i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0)
      (kx 0) (ky 0) (lenx 0) (leny 0) (temp #C(0.0 0.0)))
      (declare (type (member t nil) noconj)
        (type fixnum i info ix iy j jx jy kx ky lenx
          leny)
        (type (complex double-float) temp))
      (setf info 0)

```

```

(cond
  ((and (not (char-equal trans #\N))
        (not (char-equal trans #\T))
        (not (char-equal trans #\C)))
    (setf info 1))
  ((< m 0)
    (setf info 2))
  ((< n 0)
    (setf info 3))
  ((< lda (max (the fixnum 1) (the fixnum m)))
    (setf info 6))
  ((= incx 0)
    (setf info 8))
  ((= incy 0)
    (setf info 11)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "ZGEMV" info)
    (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
  (go end_label))
(setf noconj (char-equal trans #\T))
(cond
  ((char-equal trans #\N)
    (setf lenx n)
    (setf leny m)
    (t
      (setf lenx m)
      (setf leny n)))
  (cond
    ((> incx 0)
      (setf kx 1))
    (t
      (setf kx
        (f2cl-lib:int-sub 1
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub lenx 1)
            incx)))))
  (cond
    ((> incy 0)
      (setf ky 1))
    (t
      (setf ky
        (f2cl-lib:int-sub 1
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub leny 1)
            incy)))))
  (cond
    (

```

```

( (/ = beta one)
  (cond
    ((= incy 1)
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i leny) nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
              zero))))
        (t
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i leny) nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
              (* beta
                (f2cl-lib:fref y-%data%
                  (i)
                  ((1 *))
                  y-%offset%))))))))
    (t
      (setf iy ky)
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i leny) nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
              zero)
              (setf iy (f2cl-lib:int-add iy incy))))
        (t
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i leny) nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
              (* beta
                (f2cl-lib:fref y-%data%
                  (iy)
                  ((1 *))
                  y-%offset%))))
              (setf iy (f2cl-lib:int-add iy incy))))))))
      (if (= alpha zero) (go end_label))
    (cond
      ((char-equal trans #\N)
        (setf jx kx)
        (cond
          ((= incy 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j n) nil)
            (tagbody

```

```

(cond
  ((/= (f2c1-lib:fref x (jx) ((1 *))) zero)
    (setf temp
      (* alpha
        (f2c1-lib:fref x-%data%
          (jx)
          ((1 *))
          x-%offset%)))
    (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
      ((> i m) nil)
      (tagbody
        (setf (f2c1-lib:fref y-%data% (i) ((1 *)) y-%offset%)
          (+
            (f2c1-lib:fref y-%data%
              (i)
              ((1 *))
              y-%offset%)
            (* temp
              (f2c1-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)))))))
    (setf jx (f2c1-lib:int-add jx incx))))))
(t
  (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2c1-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (* alpha
              (f2c1-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)))
          (setf iy ky)
          (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2c1-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                (+
                  (f2c1-lib:fref y-%data%
                    (iy)
                    ((1 *))
                    y-%offset%)
                  (* temp
                    (f2c1-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))))
              (setf jx (f2c1-lib:int-add jx incx))))))
        (t
          (setf temp
            (* alpha
              (f2c1-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)))
          (setf iy ky)
          (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2c1-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                (+
                  (f2c1-lib:fref y-%data%
                    (iy)
                    ((1 *))
                    y-%offset%)
                  (* temp
                    (f2c1-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))))
              (setf jx (f2c1-lib:int-add jx incx))))))
          (setf jx (f2c1-lib:int-add jx incx))))))
    (setf jx (f2c1-lib:int-add jx incx))))))

```

```

        (setf iy (f2cl-lib:int-add iy incy))))))
      (setf jx (f2cl-lib:int-add jx incx))))))
(t
  (setf jy ky)
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf temp zero)
          (cond
            (noconj
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
                        (f2cl-lib:fref x-%data%
                          (i)
                          ((1 *))
                          x-%offset%))))))))
            (t
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2cl-lib:dconjg
                          (f2cl-lib:fref a-%data%
                            (i j)
                            ((1 lda) (1 *))
                            a-%offset%)
                        (f2cl-lib:fref x-%data%
                          (i)
                          ((1 *))
                          x-%offset%))))))))
                  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
                      (* alpha temp)))
                  (setf jy (f2cl-lib:int-add jy incy))))))
            (t
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
                (tagbody

```



```

(setf temp zero)
(setf ix kx)
(cond
  (noconj
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i m) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%))))))
    (setf ix (f2cl-lib:int-add ix incx))))
  (t
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i m) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:dconjg
                (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%))
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%))))))
        (setf ix (f2cl-lib:int-add ix incx))))))
  (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (* alpha temp)))
  (setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))

```

zgerc BLAS

— zgerc.input —

```
)set break resume
)sys rm -f zgerc.output
)spool zgerc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zgerc.help —

```
=====
zgerc examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZGERC - perform the rank 1 operation $A := \alpha x \text{conjg}(y') + A$,

SYNOPSIS

SUBROUTINE ZGERC (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)

COMPLEX*16 ALPHA

INTEGER INCX, INCY, LDA, M, N

COMPLEX*16 A(LDA, *), X(*), Y(*)

PURPOSE

ZGERC performs the rank 1 operation

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

PARAMETERS

M - INTEGER.

On entry, M specifies the number of rows of the

matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.
On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (m - 1) * \text{abs}(INCX))$. Before entry, the incremented array X must contain the m element vector x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

Y - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(INCY))$. Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, m)$. Unchanged on exit.

— zgerc.f —

SUBROUTINE ZGERC (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)

```

*   .. Scalar Arguments ..
      COMPLEX*16      ALPHA
      INTEGER         INCX, INCY, LDA, M, N
*   .. Array Arguments ..
      COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
      COMPLEX*16      ZERO
      PARAMETER       ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*   .. Local Scalars ..
      COMPLEX*16      TEMP
      INTEGER         I, INFO, IX, J, JY, KX
*   .. External Subroutines ..
      EXTERNAL        XERBLA
*   .. Intrinsic Functions ..
      INTRINSIC       DCONJG, MAX
*
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      IF ( M.LT.0 )THEN
        INFO = 1
      ELSE IF( N.LT.0 )THEN
        INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
        INFO = 5
      ELSE IF( INCY.EQ.0 )THEN
        INFO = 7
      ELSE IF( LDA.LT.MAX( 1, M ) )THEN
        INFO = 9
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZGERC ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*

```

```

      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through A.
*
      IF( INCY.GT.0 )THEN
        JY = 1
      ELSE
        JY = 1 - ( N - 1 )*INCY
      END IF
      IF( INCX.EQ.1 )THEN
        DO 20, J = 1, N
          IF( Y( JY ).NE.ZERO )THEN
            TEMP = ALPHA*DCONJG( Y( JY ) )
            DO 10, I = 1, M
              A( I, J ) = A( I, J ) + X( I )*TEMP
10            CONTINUE
            END IF
            JY = JY + INCY
          20 CONTINUE
        ELSE
          IF( INCX.GT.0 )THEN
            KX = 1
          ELSE
            KX = 1 - ( M - 1 )*INCX
          END IF
          DO 40, J = 1, N
            IF( Y( JY ).NE.ZERO )THEN
              TEMP = ALPHA*DCONJG( Y( JY ) )
              IX = KX
              DO 30, I = 1, M
                A( I, J ) = A( I, J ) + X( IX )*TEMP
                IX = IX + INCX
30              CONTINUE
            END IF
            JY = JY + INCY
          40 CONTINUE
        END IF
      RETURN
*
*      End of ZGERC .
*
      END

```

— BLAS 2 zgerc —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zgerc (m n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
              (type (complex double-float) alpha)
              (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
      ((x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%)
       (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jy kx)
                  (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info 1))
          ((< n 0)
            (setf info 2))
          ((= incx 0)
            (setf info 5))
          ((= incy 0)
            (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info 9)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "ZGERC" info)
            (go end_label)))
        (if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((> incy 0)
            (setf jy 1))
          (t
            (setf jy
              (f2cl-lib:int-sub 1
                (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                  incy)))))
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          ((> j n) nil)
              (tagbody
                (cond
                  ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
                    (setf temp

```

```

(* alpha
  (f2cl-lib:dconjg
    (f2cl-lib:fref y-%data%
      (jy)
      ((1 *))
      y-%offset%))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)
    (+
      (f2cl-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)
      (*
        (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
        temp))))))
(setf jy (f2cl-lib:int-add jy incy))))
(t
  (cond
    (> incx 0)
    (setf kx 1))
  (t
    (setf kx
      (f2cl-lib:int-sub 1
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          incx))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    (/= (f2cl-lib:fref y (jy) ((1 *))) zero)
    (setf temp
      (* alpha
        (f2cl-lib:dconjg
          (f2cl-lib:fref y-%data%
            (jy)
            ((1 *))
            y-%offset%))))
    (setf ix kx)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
        (i j)

```

```

((1 lda) (1 *))
a-%offset%)
(+
(f2cl-lib:fref a-%data%
(i j)
((1 lda) (1 *))
a-%offset%)
(*
(f2cl-lib:fref x-%data%
(ix)
((1 *))
x-%offset%)
temp)))
(setf ix (f2cl-lib:int-add ix incx))))))
(setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))))

```

zgeru BLAS

— zgeru.input —

```

)set break resume
)sys rm -f zgeru.output
)spool zgeru.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zgeru.help —

```

=====
zgeru examples
=====

=====
Man Page Details
=====

```


NAME

ZGERU - perform the rank 1 operation $A := \alpha * x * y' + A$,

SYNOPSIS

SUBROUTINE ZGERU (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)

COMPLEX*16 ALPHA

INTEGER INCX, INCY, LDA, M, N

COMPLEX*16 A(LDA, *), X(*), Y(*)

PURPOSE

ZGERU performs the rank 1 operation

where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

PARAMETERS

M - INTEGER.

On entry, M specifies the number of rows of the matrix A. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (m - 1) * \text{abs}(INCX))$. Before entry, the incremented array X must contain the m element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

Y - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(INCY))$. Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.

INCY - INTEGER.

On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on

exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m). Unchanged on exit.

— zgeru.f —

```

SUBROUTINE ZGERU ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
* .. Scalar Arguments ..
COMPLEX*16      ALPHA
INTEGER         INCX, INCY, LDA, M, N
* .. Array Arguments ..
COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16      ZERO
PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
* .. Local Scalars ..
COMPLEX*16      TEMP
INTEGER         I, INFO, IX, J, JY, KX
* .. External Subroutines ..
EXTERNAL        XERBLA
* .. Intrinsic Functions ..
INTRINSIC       MAX
*
* .. Executable Statements ..

```

```

*
*   Test the input parameters.
*
      INFO = 0
      IF      ( M.LT.0 )THEN
        INFO = 1
      ELSE IF( N.LT.0 )THEN
        INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
        INFO = 5
      ELSE IF( INCY.EQ.0 )THEN
        INFO = 7
      ELSE IF( LDA.LT.MAX( 1, M ) )THEN
        INFO = 9
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZGERU ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through A.
*
      IF( INCY.GT.0 )THEN
        JY = 1
      ELSE
        JY = 1 - ( N - 1 )*INCY
      END IF
      IF( INCX.EQ.1 )THEN
        DO 20, J = 1, N
          IF( Y( JY ).NE.ZERO )THEN
            TEMP = ALPHA*Y( JY )
            DO 10, I = 1, M
              A( I, J ) = A( I, J ) + X( I )*TEMP
10          CONTINUE
            END IF
            JY = JY + INCY
20        CONTINUE
      ELSE
        IF( INCX.GT.0 )THEN
          KX = 1
        ELSE
          KX = 1 - ( M - 1 )*INCX
        END IF
        DO 40, J = 1, N

```

```

      IF( Y( JY ).NE.ZERO )THEN
        TEMP = ALPHA*Y( JY )
        IX   = KX
        DO 30, I = 1, M
          A( I, J ) = A( I, J ) + X( IX )*TEMP
          IX         = IX       + INCX
30      CONTINUE
        END IF
        JY = JY + INCY
40     CONTINUE
      END IF
*
      RETURN
*
*   End of ZGERU .
*
      END

```

— BLAS 2 zgeru —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zgeru (m n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
              (type (complex double-float) alpha)
              (type fixnum lda incy incx n m))
    (f2cl-lib:with-multi-array-data
      ((x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%)
       (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jy 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jy kx)
                  (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info 1))
          ((< n 0)
            (setf info 2))
          ((= incx 0)
            (setf info 5))
          ((= incy 0)
            (setf info 7))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info 9)))
        (cond

```

```

( (/ = info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZGERU" info)
  (go end_label)))
(if (or (= m 0) (= n 0) (= alpha zero)) (go end_label))
(cond
  ((> incy 0)
    (setf jy 1))
  (t
    (setf jy
      (f2cl-lib:int-sub 1
        (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
          incy)))))
(cond
  ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
            (setf temp
              (* alpha
                (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%)
                  (+
                    (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%)
                    (*
                      (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                      temp))))))
              (setf jy (f2cl-lib:int-add jy incy)))))
    (t
      (cond
        ((> incx 0)
          (setf kx 1))
        (t
          (setf kx
            (f2cl-lib:int-sub 1
              (f2cl-lib:int-mul
                (f2cl-lib:int-sub m 1)
                incx)))))

```

```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref y (jy) ((1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)))
      (setf ix kx)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
          (+
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
              temp))))
        (setf ix (f2cl-lib:int-add ix incx))))))
      (setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

zhbmvm BLAS

— zhbmv.input —

```

)set break resume
)sys rm -f zhbmv.output
)spool zhbmv.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— zgbmv.help —

=====

zgbmv examples

=====

=====

Man Page Details

=====

NAME

ZGBMV - perform the matrix-vector operation $y := \alpha * A * x + \beta * y$,

SYNOPSIS

SUBROUTINE ZGBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA,
Y, INCY)

COMPLEX*16 ALPHA, BETA

INTEGER INCX, INCY, K, LDA, N

CHARACTER*1 UPLO

COMPLEX*16 A(LDA, *), X(*), Y(*)

PURPOSE

ZGBMV performs the matrix-vector operation

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian band matrix, with k super-diagonals.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the band matrix A is being supplied as follows:

UPLO = 'U' or 'u' The upper triangular part of A is being supplied.

UPLO = 'L' or 'l' The lower triangular part of A is being supplied.

Unchanged on exit.

- N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.
- K - INTEGER.
On entry, K specifies the number of super-diagonals of the matrix A. K must satisfy $0 \leq K$. Unchanged on exit.
- ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha. Unchanged on exit.
- A - COMPLEX*16 array of DIMENSION (LDA, n).

Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer the upper triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
K ), J A( M + I, J ) = matrix( I, J ) 10     CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer the lower triangular part of a hermitian band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
) A( M + I, J ) = matrix( I, J ) 10     CONTINUE 20
CONTINUE
```

Note that the imaginary parts of the diagonal ele-

ments need not be set and are assumed to be zero.
Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least ($k + 1$). Unchanged on exit.

X - COMPLEX*16 array of DIMENSION at least
($1 + (n - 1) * \text{abs}(\text{INCX})$). Before entry, the incremented array X must contain the vector x.
Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - COMPLEX*16 .

On entry, BETA specifies the scalar beta. Unchanged on exit.

Y - COMPLEX*16 array of DIMENSION at least
($1 + (n - 1) * \text{abs}(\text{INCX})$). Before entry, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

— zgbmv.f —

```

SUBROUTINE ZGBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
* .. Scalar Arguments ..
COMPLEX*16      ALPHA, BETA
INTEGER         INCX, INCY, K, LDA, N
CHARACTER*1     UPLO
* .. Array Arguments ..
COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*
*
* Level 2 Blas routine.

```

```

*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16      ONE
PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16      ZERO
PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
COMPLEX*16      TEMP1, TEMP2
INTEGER         I, INFO, IX, IY, J, JX, JY, KPLUS1, KX, KY, L
*
* .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*
* .. External Subroutines ..
EXTERNAL        XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC       DCONJG, MAX, MIN, DBLE
*
* .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ).AND.
$           .NOT.LSAME( UPLO, 'L' ) )THEN
          INFO = 1
      ELSE IF( N.LT.0 )THEN
          INFO = 2
      ELSE IF( K.LT.0 )THEN
          INFO = 3
      ELSE IF( LDA.LT.( K + 1 ) )THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 8
      ELSE IF( INCY.EQ.0 )THEN
          INFO = 11
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'ZHBMV ', INFO )
          RETURN
      END IF
*
*   Quick return if possible.
*

```

```

      IF( ( N.EQ.0 ).OR.( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$     RETURN
*
*     Set up the start points in X and Y.
*
      IF( INCX.GT.0 )THEN
        KX = 1
      ELSE
        KX = 1 - ( N - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( N - 1 )*INCY
      END IF
*
*     Start the operations. In this version the elements of the array A
*     are accessed sequentially with one pass through A.
*
*     First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, N
              Y( I ) = BETA*Y( I )
20          CONTINUE
            END IF
          ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
              DO 30, I = 1, N
                Y( IY ) = ZERO
                IY = IY + INCY
30          CONTINUE
            ELSE
              DO 40, I = 1, N
                Y( IY ) = BETA*Y( IY )
                IY = IY + INCY
40          CONTINUE
            END IF
          END IF
        END IF
        IF( ALPHA.EQ.ZERO )
$       RETURN
        IF( LSAME( UPLO, 'U' ) )THEN

```

```

*
*      Form y when upper triangle of A is stored.
*
      KPLUS1 = K + 1
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          L      = KPLUS1 - J
          DO 50, I = MAX( 1, J - K ), J - 1
            Y( I ) = Y( I ) + TEMP1*A( L + I, J )
            TEMP2 = TEMP2 + DCONJG( A( L + I, J ) ) * X( I )
50          CONTINUE
          Y( J ) = Y( J ) + TEMP1*DBLE( A( KPLUS1, J ) )
          $      + ALPHA*TEMP2
60        CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          IX = KX
          IY = KY
          L = KPLUS1 - J
          DO 70, I = MAX( 1, J - K ), J - 1
            Y( IY ) = Y( IY ) + TEMP1*A( L + I, J )
            TEMP2 = TEMP2 + DCONJG( A( L + I, J ) ) * X( IX )
            IX = IX + INCX
            IY = IY + INCY
70          CONTINUE
          Y( JY ) = Y( JY ) + TEMP1*DBLE( A( KPLUS1, J ) )
          $      + ALPHA*TEMP2
          JX = JX + INCX
          JY = JY + INCY
          IF( J.GT.K )THEN
            KX = KX + INCX
            KY = KY + INCY
          END IF
80        CONTINUE
      END IF
    ELSE
*
*      Form y when lower triangle of A is stored.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 100, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          Y( J ) = Y( J ) + TEMP1*DBLE( A( 1, J ) )

```

```

          L      = 1      - J
          DO 90, I = J + 1, MIN( N, J + K )
              Y( I ) = Y( I ) + TEMP1*A( L + I, J )
              TEMP2 = TEMP2 + DCONJG( A( L + I, J ) ) * X( I )
90          CONTINUE
          Y( J ) = Y( J ) + ALPHA*TEMP2
100         CONTINUE
          ELSE
              JX = KX
              JY = KY
              DO 120, J = 1, N
                  TEMP1 = ALPHA*X( JX )
                  TEMP2 = ZERO
                  Y( JY ) = Y( JY ) + TEMP1*DBLE( A( 1, J ) )
                  L      = 1      - J
                  IX     = JX
                  IY     = JY
                  DO 110, I = J + 1, MIN( N, J + K )
                      IX = IX + INCX
                      IY = IY + INCY
                      Y( IY ) = Y( IY ) + TEMP1*A( L + I, J )
                      TEMP2 = TEMP2 + DCONJG( A( L + I, J ) ) * X( IX )
110                 CONTINUE
                  Y( JY ) = Y( JY ) + ALPHA*TEMP2
                  JX = JX + INCX
                  JY = JY + INCY
120             CONTINUE
          END IF
      END IF
*
      RETURN
*
*      End of ZHBMV .
*
      END

```

— BLAS 2 zhbmv —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhbmv (uplo n k alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
              (type (complex double-float) beta alpha)
              (type fixnum incx incy lda k n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data

```

```

((uplo character uplo-%data% uplo-%offset%)
(a (complex double-float) a-%data% a-%offset%)
(x (complex double-float) x-%data% x-%offset%)
(y (complex double-float) y-%data% y-%offset%))
(prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kplus1 0) (kx 0)
      (ky 0) (l 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
(declare (type fixnum i info ix iy j jx jy kplus1 kx ky l)
          (type (complex double-float) temp1 temp2))
(setf info 0)
(cond
  ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
   (setf info 1))
  ((< n 0)
   (setf info 2))
  ((< k 0)
   (setf info 3))
  ((< lda (f2cl-lib:int-add k 1))
   (setf info 6))
  ((= incx 0)
   (setf info 8))
  ((= incy 0)
   (setf info 11)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZHEBMV" info)
   (go end_label)))
(if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
    (f2cl-lib:int-sub 1
     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                       incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
    (f2cl-lib:int-sub 1
     (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                       incy)))))
(cond
  ((/= beta one)
   (cond
    ((= incy 1)
     (cond

```

```

(= beta zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
      zero))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
        (* beta
          (f2cl-lib:fref y-%data%
            (i)
            ((1 *))
            y-%offset%)))))))))
(t
  (setf iy ky)
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
            zero)
            (setf iy (f2cl-lib:int-add iy incy)))))
    (t
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
            (* beta
              (f2cl-lib:fref y-%data%
                (iy)
                ((1 *))
                y-%offset%)))
            (setf iy (f2cl-lib:int-add iy incy)))))
    (if (= alpha zero) (go end_label))
  (cond
    ((char-equal uplo #\U)
      (setf kplus1 (f2cl-lib:int-add k 1))
      (cond
        ((and (= incx 1) (= incy 1))
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            (> j n) nil)
            (tagbody
              (setf temp1
                (* alpha
                  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
              (setf temp2 zero)

```

```

(setf l (f2cl-lib:int-sub kplus1 j))
(f2cl-lib:fdo (i
  (max (the fixnum 1)
        (the fixnum
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub
              k))))))
  (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i) j)
          ((1 lda) (1 *))
          a-%offset%))))))
  (setf temp2
    (+ temp2
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add 1 i) j)
            ((1 lda) (1 *))
            a-%offset%))
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))))
  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
      (* temp1
        (coerce (realpart
          (f2cl-lib:fref a-%data%
            (kplus1 j)
            ((1 lda) (1 *))
            a-%offset%)) 'double-float))
        (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))

```



```

(setf temp2 zero)
(setf ix kx)
(setf iy ky)
(setf l (f2cl-lib:int-sub kplus1 j))
(f2cl-lib:fdo (i
  (max (the fixnum 1)
        (the fixnum
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub
              k))))))
  (f2cl-lib:int-add i 1))
  ((> i
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
   nil)
  (tagbody
    (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (+
        (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add l i) j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              ((f2cl-lib:int-add l i) j)
              ((1 lda) (1 *))
              a-%offset%))
          (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%))))))
    (setf ix (f2cl-lib:int-add ix incx))
    (setf iy (f2cl-lib:int-add iy incy))))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* temp1
      (coerce (realpart
        (f2cl-lib:fref a-%data%
          (kplus1 j)
          ((1 lda) (1 *))
          a-%offset%)) 'double-float))
      (* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(cond
  ((> j k)

```

```

      (setf kx (f2cl-lib:int-add kx incx))
      (setf ky (f2cl-lib:int-add ky incy)))))))))
(t
 (cond
  ((and (= incx 1) (= incy 1))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
   (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
    (setf temp2 zero)
    (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
        (* temp1
          (coerce (realpart
                    (f2cl-lib:fref a-%data%
                                   (1 j)
                                   ((1 lda) (1 *))
                                   a-%offset%)) 'double-float))))
    (setf l (f2cl-lib:int-sub 1 j))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
                  ((> i
                    (min (the fixnum n)
                        (the fixnum
                          (f2cl-lib:int-add j k)))))
                  nil)
    (tagbody
     (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
      (+
        (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add 1 i) j)
                        ((1 lda) (1 *))
                        a-%offset%))))
     (setf temp2
      (+ temp2
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i) j)
                          ((1 lda) (1 *))
                          a-%offset%))
          (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%))))))
    (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)

```

```

      (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
         (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
      (setf temp2 zero)
      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
          (* temp1
            (coerce (realpart
              (f2cl-lib:fref a-%data%
                (1 j)
                ((1 lda) (1 *))
                a-%offset%)) 'double-float))))))
      (setf l (f2cl-lib:int-sub 1 j))
      (setf ix jx)
      (setf iy jy)
      (f2cl-lib:fd0 (i (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add i 1))
        ((> i
          (min (the fixnum n)
            (the fixnum
              (f2cl-lib:int-add j k))))
          nil)
        (tagbody
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
            (+
              (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
              (* temp1
                (f2cl-lib:fref a-%data%
                  ((f2cl-lib:int-add l i) j)
                  ((1 lda) (1 *))
                  a-%offset%))))))
          (setf temp2
            (+ temp2
              (*
                (f2cl-lib:dconjg
                  (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add l i) j)
                    ((1 lda) (1 *))
                    a-%offset%))
                (f2cl-lib:fref x-%data%

```

```

                                (ix)
                                ((1 *))
                                x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%
      (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%
          (* alpha temp2))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))

```

zhemv BLAS

— zhemv.input —

```

)set break resume
)sys rm -f zhemv.output
)spool zhemv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhemv.help —

```

=====
zhemv examples
=====

=====
Man Page Details
=====

NAME
    ZHEMV - perform the matrix-vector operation    y := alpha*A*x
    + beta*y,

SYNOPSIS
    SUBROUTINE ZHEMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y,

```

```

                INCY )

COMPLEX*16     ALPHA, BETA

INTEGER        INCX, INCY, LDA, N

CHARACTER*1    UPLO

COMPLEX*16     A( LDA, * ), X( * ), Y( * )

```

PURPOSE

ZHEMV performs the matrix-vector operation

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix.

PARAMETERS

UPLO - CHARACTER*1.
On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements need not be set and

are assumed to be zero. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCY}))$. Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

— zhemv.f —

```

SUBROUTINE ZHEMV ( UPLO, N, ALPHA, A, LDA, X, INCX,
$                BETA, Y, INCY )
*   .. Scalar Arguments ..
COMPLEX*16      ALPHA, BETA
INTEGER         INCX, INCY, LDA, N
CHARACTER*1     UPLO
*   .. Array Arguments ..
COMPLEX*16      A( LDA, * ), X( * ), Y( * )
*
*   ..
*
*   Level 2 Blas routine.
*
```

```

* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
COMPLEX*16      ONE
PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16      ZERO
PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*   .. Local Scalars ..
COMPLEX*16      TEMP1, TEMP2
INTEGER         I, INFO, IX, IY, J, JX, JY, KX, KY
*
*   .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*
*   .. External Subroutines ..
EXTERNAL        XERBLA
*
*   .. Intrinsic Functions ..
INTRINSIC       DCONJG, MAX, DBLE
*
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$           .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = 1
      ELSE IF( N.LT.0 ) THEN
          INFO = 2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = 5
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 7
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 10
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZHEMV ', INFO )
          RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ) .OR.( ( ALPHA.EQ.ZERO ) .AND.( BETA.EQ.ONE ) ) )
$      RETURN
*

```

```

*      Set up the start points in X and Y.
*
      IF( INCX.GT.0 )THEN
        KX = 1
      ELSE
        KX = 1 - ( N - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( N - 1 )*INCY
      END IF
*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through the triangular part
*      of A.
*
*      First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, N
              Y( I ) = BETA*Y( I )
20          CONTINUE
          END IF
        ELSE
          IY = KY
          IF( BETA.EQ.ZERO )THEN
            DO 30, I = 1, N
              Y( IY ) = ZERO
              IY = IY + INCY
30          CONTINUE
          ELSE
            DO 40, I = 1, N
              Y( IY ) = BETA*Y( IY )
              IY = IY + INCY
40          CONTINUE
          END IF
        END IF
      END IF
      IF( ALPHA.EQ.ZERO )
        $ RETURN
      IF( LSAME( UPLO, 'U' ) )THEN
*
*      Form y when A is stored in upper triangle.

```



```

*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          DO 50, I = 1, J - 1
            Y( I ) = Y( I ) + TEMP1*A( I, J )
            TEMP2 = TEMP2 + DCONJG( A( I, J ) )*X( I )
50          CONTINUE
          Y( J ) = Y( J ) + TEMP1*DBLE( A( J, J ) ) + ALPHA*TEMP2
60        CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          IX = KX
          IY = KY
          DO 70, I = 1, J - 1
            Y( IY ) = Y( IY ) + TEMP1*A( I, J )
            TEMP2 = TEMP2 + DCONJG( A( I, J ) )*X( IX )
            IX = IX + INCX
            IY = IY + INCY
70          CONTINUE
          Y( JY ) = Y( JY ) + TEMP1*DBLE( A( J, J ) ) + ALPHA*TEMP2
          JX = JX + INCX
          JY = JY + INCY
80        CONTINUE
      END IF
    ELSE
*
*      Form y when A is stored in lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 100, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          Y( J ) = Y( J ) + TEMP1*DBLE( A( J, J ) )
          DO 90, I = J + 1, N
            Y( I ) = Y( I ) + TEMP1*A( I, J )
            TEMP2 = TEMP2 + DCONJG( A( I, J ) )*X( I )
90          CONTINUE
          Y( J ) = Y( J ) + ALPHA*TEMP2
100         CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 120, J = 1, N
          TEMP1 = ALPHA*X( JX )

```

```

      TEMP2  = ZERO
      Y( JY ) = Y( JY ) + TEMP1*DBLE( A( J, J ) )
      IX     = JX
      IY     = JY
      DO 110, I = J + 1, N
        IX   = IX   + INCX
        IY   = IY   + INCY
        Y( IY ) = Y( IY ) + TEMP1*A( I, J )
        TEMP2  = TEMP2  + DCONJG( A( I, J ) ) * X( IX )
110      CONTINUE
        Y( JY ) = Y( JY ) + ALPHA*TEMP2
        JX     = JX   + INCX
        JY     = JY   + INCY
120      CONTINUE
      END IF
    END IF
*
    RETURN
*
*   End of ZHEMV .
*
    END

```

— BLAS 2 zhemv —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zhemv (uplo n alpha a lda x incx beta y incy)
    (declare (type (simple-array (complex double-float) (*)) y x a)
      (type (complex double-float) beta alpha)
      (type fixnum incy incx lda n)
      (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
        (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy kx ky)
          (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          ((< n 0)

```

```

    (setf info 2))
  (< lda (max (the fixnum 1) (the fixnum n)))
  (setf info 5))
  ((= incx 0)
   (setf info 7))
  ((= incy 0)
   (setf info 10)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZHEMV" info)
   (go end_label)))
(if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
(cond
  ((> incx 0)
   (setf kx 1))
  (t
   (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx)))))
(cond
  ((> incy 0)
   (setf ky 1))
  (t
   (setf ky
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incy)))))
(cond
  ((/= beta one)
   (cond
    ((= incy 1)
     (cond
      ((= beta zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
         (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
           zero))))
      (t
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
         (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
           (* beta
            (f2cl-lib:fref y-%data%
              (i)
              ((1 *))

```

```

                                y-%offset%)))))))))
(t
  (setf iy ky)
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
            zero)
          (setf iy (f2cl-lib:int-add iy incy))))))
    (t
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
            (* beta
              (f2cl-lib:fref y-%data%
                (iy)
                ((1 *))
                y-%offset%)))
          (setf iy (f2cl-lib:int-add iy incy)))))))))
(if (= alpha zero) (go end_label))
(cond
  ((char-equal uplo #\U)
    (cond
      ((and (= incx 1) (= incy 1))
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j n) nil)
          (tagbody
            (setf temp1
              (* alpha
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
            (setf temp2 zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i
                (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                nil)
              (tagbody
                (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                  (+
                    (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                    (* temp1
                      (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%))))
                (setf temp2
                  (+ temp2
                    (*

```

```

(f2cl-lib:dconjg
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%))
(f2cl-lib:fref x-%data%
  (i)
  ((1 *))
  x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (* temp1
      (coerce (realpart
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)) 'double-float))
      (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
      (setf temp2 zero)
      (setf ix kx)
      (setf iy ky)
      (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
        ((> i
          (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
          nil)
          (tagbody
            (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
              (+
                (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                (* temp1
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%))))))
            (setf temp2
              (+ temp2
                (*
                  (f2cl-lib:dconjg
                    (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))))

```

```

                                a-%offset%))
(f2cl-lib:fref x-%data%
  (ix)
  ((1 *))
  x-%offset%))))
(setf ix (f2cl-lib:int-add ix incx))
(setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* temp1
      (coerce (realpart
                (f2cl-lib:fref a-%data%
                              (j j)
                              ((1 lda) (1 *))
                              a-%offset%)) 'double-float)))
      (* alpha temp2))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp1
          (* alpha
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
        (setf temp2 zero)
        (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
          (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
            (* temp1
              (coerce (realpart
                        (f2cl-lib:fref a-%data%
                                      (j j)
                                      ((1 lda) (1 *))
                                      a-%offset%)) 'double-float))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
              (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                (+
                  (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
                  (* temp1
                    (f2cl-lib:fref a-%data%
                                      (i j)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
                (setf temp2
                  (+ temp2

```

```

(*
  (f2cl-lib:dconjg
    (f2cl-lib:fref a-%data%
      (i j)
      ((1 lda) (1 *))
      a-%offset%))
    (f2cl-lib:fref x-%data%
      (i)
      ((1 *))
      x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
        (* alpha temp2))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
      (setf temp2 zero)
      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
              (* temp1
                (coerce (realpart
                  (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%)) 'double-float))))))
      (setf ix jx)
      (setf iy jy)
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))
          (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                (+
                  (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
                  (* temp1
                    (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))))
          (setf temp2
            (+ temp2
              (*

```

```

(f2cl-lib:dconjg
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%))
(f2cl-lib:fref x-%data%
  (ix)
  ((1 *))
  x-%offset%))))))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil))))))

```

zher2 BLAS

— zher2.input —

```

)set break resume
)sys rm -f zher2.output
)spool zher2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zher2.help —

```

=====
zher2 examples
=====

```

```

=====
Man Page Details
=====

```


NAME

ZHER2 - perform the hermitian rank 2 operation $A := \alpha x x^* + \text{conjg}(\alpha) y y^* + A$,

SYNOPSIS

```
SUBROUTINE ZHER2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA
                  )
```

```
COMPLEX*16  ALPHA
```

```
INTEGER      INCX, INCY, LDA, N
```

```
CHARACTER*1  UPLO
```

```
COMPLEX*16  A( LDA, * ), X( * ), Y( * )
```

PURPOSE

ZHER2 performs the hermitian rank 2 operation

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

Y - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCY}))$. Before entry, the incremented array Y must contain the n element vector y. Unchanged on exit.

INCY - INTEGER.
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

— zher2.f —

```

SUBROUTINE ZHER2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )
* .. Scalar Arguments ..
COMPLEX*16      ALPHA
INTEGER         INCX, INCY, LDA, N
CHARACTER*1     UPLO
* .. Array Arguments ..

```

```

      COMPLEX*16      A( LDA, * ), X( * ), Y( * )
      ..
*
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16      ZERO
PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
COMPLEX*16      TEMP1, TEMP2
INTEGER         I, INFO, IX, IY, J, JX, JY, KX, KY
*
* .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*
* .. External Subroutines ..
EXTERNAL        XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC       DCONJG, MAX, DBLE
*
* ..
*
* .. Executable Statements ..
*
* Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$           .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = 1
      ELSE IF( N.LT.0 ) THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 5
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 7
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = 9
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZHER2 ', INFO )
          RETURN
      END IF
*
* Quick return if possible.
*

```

```

      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$     RETURN
*
*     Set up the start points in X and Y if the increments are not both
*     unity.
*
      IF( ( INCX.NE.1 ).OR.( INCY.NE.1 ) )THEN
        IF( INCX.GT.0 )THEN
          KX = 1
        ELSE
          KX = 1 - ( N - 1 )*INCX
        END IF
        IF( INCY.GT.0 )THEN
          KY = 1
        ELSE
          KY = 1 - ( N - 1 )*INCY
        END IF
        JX = KX
        JY = KY
      END IF
*
*     Start the operations. In this version the elements of A are
*     accessed sequentially with one pass through the triangular part
*     of A.
*
      IF( LSAME( UPLO, 'U' ) )THEN
*
*       Form A when A is stored in the upper triangle.
*
        IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
          DO 20, J = 1, N
            IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
              TEMP1 = ALPHA*DCONJG( Y( J ) )
              TEMP2 = DCONJG( ALPHA*X( J ) )
              DO 10, I = 1, J - 1
                A( I, J ) = A( I, J ) + X( I )*TEMP1 + Y( I )*TEMP2
10             CONTINUE
                A( J, J ) = DBLE( A( J, J ) ) +
$                 DBLE( X( J )*TEMP1 + Y( J )*TEMP2 )
              ELSE
                A( J, J ) = DBLE( A( J, J ) )
              END IF
20             CONTINUE
            ELSE
              DO 40, J = 1, N
                IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                  TEMP1 = ALPHA*DCONJG( Y( JY ) )
                  TEMP2 = DCONJG( ALPHA*X( JX ) )
                  IX = KX
                  IY = KY

```

```

      DO 30, I = 1, J - 1
        A( I, J ) = A( I, J ) + X( IX )*TEMP1
          $                                     + Y( IY )*TEMP2
          IX      = IX      + INCX
          IY      = IY      + INCY
30      CONTINUE
        A( J, J ) = DBLE( A( J, J ) ) +
          $                                     DBLE( X( JX )*TEMP1 + Y( JY )*TEMP2 )
        ELSE
          A( J, J ) = DBLE( A( J, J ) )
        END IF
        JX = JX + INCX
        JY = JY + INCY
40      CONTINUE
      END IF
    ELSE
*
*      Form A when A is stored in the lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
          IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
            TEMP1 = ALPHA*DCONJG( Y( J ) )
            TEMP2 = DCONJG( ALPHA*X( J ) )
            A( J, J ) = DBLE( A( J, J ) ) +
              $                                     DBLE( X( J )*TEMP1 + Y( J )*TEMP2 )
            DO 50, I = J + 1, N
              A( I, J ) = A( I, J ) + X( I )*TEMP1 + Y( I )*TEMP2
50          CONTINUE
            ELSE
              A( J, J ) = DBLE( A( J, J ) )
            END IF
60          CONTINUE
        ELSE
          DO 80, J = 1, N
            IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
              TEMP1 = ALPHA*DCONJG( Y( JY ) )
              TEMP2 = DCONJG( ALPHA*X( JX ) )
              A( J, J ) = DBLE( A( J, J ) ) +
                $                                     DBLE( X( JX )*TEMP1 + Y( JY )*TEMP2 )
              IX = JX
              IY = JY
              DO 70, I = J + 1, N
                IX = IX + INCX
                IY = IY + INCY
                A( I, J ) = A( I, J ) + X( IX )*TEMP1
              $                                     + Y( IY )*TEMP2
70          CONTINUE
            ELSE
              A( J, J ) = DBLE( A( J, J ) )

```

```

        END IF
        JX = JX + INCX
        JY = JY + INCY
80      CONTINUE
      END IF
    END IF
*
    RETURN
*
*    End of ZHER2 .
*
    END

```

— BLAS 2 zher2 —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zher2 (uplo n alpha x incx y incy a lda)
    (declare (type (simple-array (complex double-float) (*)) a y x)
              (type (complex double-float) alpha)
              (type fixnum lda incy incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%)
       (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (kx 0) (ky 0)
             (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy kx ky)
                  (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((< n 0)
          (setf info 2))
         ((= incx 0)
          (setf info 5))
         ((= incy 0)
          (setf info 7))
         ((< lda (max (the fixnum 1) (the fixnum n)))
          (setf info 9)))
        (cond
         ((/= info 0)
          (error

```

```

" ** On entry to ~a parameter number ~a had an illegal value~%"
"ZHER2" info)
(go end_label)))
(if (or (= n 0) (= alpha zero)) (go end_label))
(cond
  ((or (/= incx 1) (/= incy 1))
   (cond
    ((> incx 0)
     (setf kx 1))
    (t
     (setf kx
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub n 1)
                              incx))))))
   (cond
    ((> incy 0)
     (setf ky 1))
    (t
     (setf ky
           (f2cl-lib:int-sub 1
                             (f2cl-lib:int-mul
                              (f2cl-lib:int-sub n 1)
                              incy))))))
   (setf jx kx)
   (setf jy ky)))
(cond
  ((char-equal uplo #\U)
   (cond
    ((and (= incx 1) (= incy 1))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
     (tagbody
      (cond
       ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
              (/= (f2cl-lib:fref y (j) ((1 *))) zero))
        (setf temp1
              (* alpha
                (f2cl-lib:dconjg
                 (f2cl-lib:fref y-%data%
                               (j)
                               ((1 *))
                               y-%offset%))))))
        (setf temp2
              (coerce
               (f2cl-lib:dconjg
                (* alpha
                  (f2cl-lib:fref x-%data%
                                (j)
                                ((1 *))

```

```

                                x-%offset%)))
      '(complex double-float)))
(f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
  ((> i
    (f2c1-lib:int-add j
      (f2c1-lib:int-sub 1)))
    nil)
(tagbody
  (setf (f2c1-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)
    (+
      (f2c1-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)
      (*
        (f2c1-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)
        temp1)
      (*
        (f2c1-lib:fref y-%data%
          (i)
          ((1 *))
          y-%offset%)
        temp2))))))
(setf (f2c1-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)
  (coerce
    (+
      (coerce (realpart
        (f2c1-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)) 'double-float)
      (coerce (realpart
        (+
          (*
            (f2c1-lib:fref x-%data%
              (j)
              ((1 *))
              x-%offset%)
            temp1)
          (*
            (f2c1-lib:fref y-%data%
              (j)
              ((1 *))
              y-%offset%)
            temp2)
          )
        ) 'double-float)
    'double-float))

```



```

                                (j)
                                ((1 *))
                                y-%offset%)
                                temp2))) 'double-float))
                                '(complex double-float)))
(t
  (setf (f2cl-lib:fref a-%data%
                      (j j)
                      ((1 lda) (1 *))
                      a-%offset%)
        (coerce
         (coerce (realpart
                  (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%)) 'double-float)
          '(complex double-float))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
    (tagbody
      (cond
        ((or (/= (f2cl-lib:fref x (jx) ((1 *)) zero)
                  (/= (f2cl-lib:fref y (jy) ((1 *)) zero))
              (setf temp1
                    (* alpha
                      (f2cl-lib:dconjg
                       (f2cl-lib:fref y-%data%
                                       (jy)
                                       ((1 *))
                                       y-%offset%))))
              (setf temp2
                    (coerce
                     (f2cl-lib:dconjg
                      (* alpha
                        (f2cl-lib:fref x-%data%
                                       (jx)
                                       ((1 *))
                                       x-%offset%)))
                     '(complex double-float)))
              (setf ix kx)
              (setf iy ky)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            ((> i
                                (f2cl-lib:int-add j
                                  (f2cl-lib:int-sub 1)))
                              nil)
                (tagbody
                  (setf (f2cl-lib:fref a-%data%
                                      (i j)

```

```

((1 lda) (1 *))
a-%offset%)

(+
(f2cl-lib:fref a-%data%
  (i j)
  ((1 lda) (1 *))
  a-%offset%)

(*
(f2cl-lib:fref x-%data%
  (ix)
  ((1 *))
  x-%offset%)

temp1)
(*
(f2cl-lib:fref y-%data%
  (iy)
  ((1 *))
  y-%offset%)

temp2)))
(setf ix (f2cl-lib:int-add ix incx))
(setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)

(coerce
(+
(coerce (realpart
(f2cl-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)) 'double-float)
(coerce (realpart
(+
(*
(f2cl-lib:fref x-%data%
  (jx)
  ((1 *))
  x-%offset%)

temp1)
(*
(f2cl-lib:fref y-%data%
  (jy)
  ((1 *))
  y-%offset%)

temp2))) 'double-float)))
'(complex double-float)))

(t
(setf (f2cl-lib:fref a-%data%
  (j j)

```

```

((1 lda) (1 *))
a-%offset%)

(coerce
(coerce (realpart
(f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)) 'double-float)
'(complex double-float))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))))))
(t
(cond
((and (= incx 1) (= incy 1))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(cond
((or (/= (f2cl-lib:fref x (j) ((1 *)) zero)
(/= (f2cl-lib:fref y (j) ((1 *)) zero))
(setf temp1
(* alpha
(f2cl-lib:dconjg
(f2cl-lib:fref y-%data%
(j)
((1 *))
y-%offset%))))
(setf temp2
(coerce
(f2cl-lib:dconjg
(* alpha
(f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)))
'(complex double-float)))
(setf (f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)
(coerce
(+
(coerce (realpart
(f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)) 'double-float)
(coerce (realpart
(+
(*

```

```

(f2c1-lib:fref x-%data%
  (j)
  ((1 *))
  x-%offset%)
temp1)
(*
(f2c1-lib:fref y-%data%
  (j)
  ((1 *))
  y-%offset%)
temp2))) 'double-float))
'(complex double-float)))
(f2c1-lib:fdo (i (f2c1-lib:int-add j 1)
  (f2c1-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2c1-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)
    (+
      (f2c1-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)
      (*
        (f2c1-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)
        temp1)
      (*
        (f2c1-lib:fref y-%data%
          (i)
          ((1 *))
          y-%offset%)
        temp2))))))
(t
  (setf (f2c1-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)
    (coerce
      (coerce (realpart
        (f2c1-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%) 'double-float)
        '(complex double-float))))))
(t

```

```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
      (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
      (setf temp1
        (* alpha
          (f2cl-lib:dconjg
            (f2cl-lib:fref y-%data%
              (jy)
              ((1 *))
              y-%offset%))))))
    (setf temp2
      (coerce
        (f2cl-lib:dconjg
          (* alpha
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%))))
        '(complex double-float)))
    (setf (f2cl-lib:fref a-%data%
      (j j)
      ((1 lda) (1 *))
      a-%offset%)
      (coerce
        (+
          (coerce (realpart
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%)) 'double-float)
          (coerce (realpart
            (+
              (*
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%)
                temp1)
              (*
                (f2cl-lib:fref y-%data%
                  (jy)
                  ((1 *))
                  y-%offset%)
                temp2))) 'double-float))
          '(complex double-float)))
      (setf ix jx)
      (setf iy jy)

```

```

(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
               (f2cl-lib:int-add i 1))
              (> i n) nil)
(tagbody
 (setf ix (f2cl-lib:int-add ix incx))
 (setf iy (f2cl-lib:int-add iy incy))
 (setf (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%)
        (+
         (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)
         (*
          (f2cl-lib:fref x-%data%
                         (ix)
                         ((1 *))
                         x-%offset%)
          temp1)
         (*
          (f2cl-lib:fref y-%data%
                         (iy)
                         ((1 *))
                         y-%offset%)
          temp2))))))
(t
 (setf (f2cl-lib:fref a-%data%
                     (j j)
                     ((1 lda) (1 *))
                     a-%offset%)
       (coerce
        (coerce (realpart
                  (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%)) 'double-float)
        '(complex double-float))))
 (setf jx (f2cl-lib:int-add jx incx))
 (setf jy (f2cl-lib:int-add jy incy))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

zher BLAS**— zher.input —**

```

)set break resume
)sys rm -f zher.output
)spool zher.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zher.help —

```

=====
zher examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZHER - perform the hermitian rank 1 operation $A := \alpha * x * \text{conjg}(x') + A$,

SYNOPSIS

SUBROUTINE ZHER (UPLO, N, ALPHA, X, INCX, A, LDA)

DOUBLE PRECISION ALPHA

INTEGER INCX, LDA, N

CHARACTER*1 UPLO

COMPLEX*16 A(LDA, *), X(*)

PURPOSE

ZHER performs the hermitian rank 1 operation

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix.

PARAMETERS

UPLO - CHARACTER*1.
On entry, UPLO specifies whether the upper or lower triangular part of the array A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of A is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of A is to be referenced.

Unchanged on exit.

N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDA - INTEGER.
 On entry, LDA specifies the first dimension of A as
 declared in the calling (sub) program. LDA must be at
 least $\max(1, n)$. Unchanged on exit.

— zher.f —

```

SUBROUTINE ZHER ( UPLO, N, ALPHA, X, INCX, A, LDA )
*
* .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, LDA, N
CHARACTER*1       UPLO
*
* .. Array Arguments ..
COMPLEX*16        A( LDA, * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16        ZERO
PARAMETER         ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
COMPLEX*16        TEMP
INTEGER           I, INFO, IX, J, JX, KX
*
* .. External Functions ..
LOGICAL           LSAME
EXTERNAL          LSAME
*
* .. External Subroutines ..
EXTERNAL          XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC         DCONJG, MAX, DBLE
*
* ..
*
* .. Executable Statements ..
*
* Test the input parameters.
*
*
  INFO = 0
  IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$         .NOT.LSAME( UPLO, 'L' ) ) THEN

```

```

        INFO = 1
      ELSE IF( N.LT.0 )THEN
        INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
        INFO = 5
      ELSE IF( LDA.LT.MAX( 1, N ) )THEN
        INFO = 7
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZHER ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.DBLE( ZERO ) ) )
$    RETURN
*
*   Set the start point in X if the increment is not unity.
*
      IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
        KX = 1
      END IF
*
*   Start the operations. In this version the elements of A are
*   accessed sequentially with one pass through the triangular part
*   of A.
*
      IF( LSAME( UPLO, 'U' ) )THEN
*
*       Form A when A is stored in upper triangle.
*
        IF( INCX.EQ.1 )THEN
          DO 20, J = 1, N
            IF( X( J ).NE.ZERO )THEN
              TEMP = ALPHA*DCONJG( X( J ) )
              DO 10, I = 1, J - 1
                A( I, J ) = A( I, J ) + X( I )*TEMP
10              CONTINUE
                A( J, J ) = DBLE( A( J, J ) ) + DBLE( X( J )*TEMP )
              ELSE
                A( J, J ) = DBLE( A( J, J ) )
              END IF
20            CONTINUE
          ELSE
            JX = KX
            DO 40, J = 1, N
              IF( X( JX ).NE.ZERO )THEN

```

```

        TEMP = ALPHA*DCONJG( X( JX ) )
        IX   = KX
        DO 30, I = 1, J - 1
            A( I, J ) = A( I, J ) + X( IX )*TEMP
            IX         = IX         + INCX
30      CONTINUE
        A( J, J ) = DBLE( A( J, J ) ) + DBLE( X( JX )*TEMP )
        ELSE
            A( J, J ) = DBLE( A( J, J ) )
        END IF
        JX = JX + INCX
40      CONTINUE
    END IF
ELSE
*
*      Form A when A is stored in lower triangle.
*
    IF( INCX.EQ.1 )THEN
        DO 60, J = 1, N
            IF( X( J ).NE.ZERO )THEN
                TEMP = ALPHA*DCONJG( X( J ) )
                A( J, J ) = DBLE( A( J, J ) ) + DBLE( TEMP*X( J ) )
                DO 50, I = J + 1, N
                    A( I, J ) = A( I, J ) + X( I )*TEMP
50              CONTINUE
                ELSE
                    A( J, J ) = DBLE( A( J, J ) )
                END IF
60          CONTINUE
            ELSE
                JX = KX
                DO 80, J = 1, N
                    IF( X( JX ).NE.ZERO )THEN
                        TEMP = ALPHA*DCONJG( X( JX ) )
                        A( J, J ) = DBLE( A( J, J ) ) + DBLE( TEMP*X( JX ) )
                        IX = JX
                        DO 70, I = J + 1, N
                            IX = IX + INCX
                            A( I, J ) = A( I, J ) + X( IX )*TEMP
70                        CONTINUE
                        ELSE
                            A( J, J ) = DBLE( A( J, J ) )
                        END IF
                        JX = JX + INCX
80                    CONTINUE
                END IF
            END IF
*
            RETURN
*

```

```

*      End of ZHER  .
*
      END

```

— BLAS 2 zher —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zher (uplo n alpha x incx a lda)
    (declare (type (simple-array (complex double-float) (*)) a x)
              (type (double-float) alpha)
              (type fixnum lda incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (a (complex double-float) a-%data% a-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jx kx)
                  (type (complex double-float) temp))
        (setf info 0)
        (cond
         ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
         ((< n 0)
          (setf info 2))
         ((= incx 0)
          (setf info 5))
         ((< lda (max (the fixnum 1) (the fixnum n)))
          (setf info 7)))
        (cond
         ((/= info 0)
          (error
           " ** On entry to ~a parameter number ~a had an illegal value~%"
           "ZHER" info)
          (go end_label)))
        (if (or (= n 0) (= alpha (coerce (realpart zero) 'double-float)))
            (go end_label))
        (cond
         ((<= incx 0)
          (setf kx
                (f2cl-lib:int-sub 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                                       incx))))
         ((/= incx 1)
          (setf kx 1)))

```

```
(cond
((char-equal uplo #\U)
(cond
(= incx 1)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(cond
(/= (f2cl-lib:fref x (j) ((1 *))) zero)
(setf temp
(coerce
(* alpha
(f2cl-lib:dconjg
(f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)))
'complex double-float)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i
(f2cl-lib:int-add j
(f2cl-lib:int-sub 1)))
nil)
(tagbody
(setf (f2cl-lib:fref a-%data%
(i j)
((1 lda) (1 *))
a-%offset%)
(+
(f2cl-lib:fref a-%data%
(i j)
((1 lda) (1 *))
a-%offset%)
(*
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
temp))))))
(setf (f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)
(coerce
(+
(coerce realpart
(f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)) 'double-float)
```

```

(coerce (realpart
  (*
    (f2cl-lib:fref x-%data%
      (j)
      ((1 *))
      x-%offset%)
    temp)) 'double-float))
'(complex double-float)))
(t
  (setf (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)
    (coerce
      (coerce (realpart
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)) 'double-float)
      '(complex double-float))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (coerce
              (* alpha
                (f2cl-lib:dconjg
                  (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)))
              '(complex double-float)))
          (setf ix kx)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub 1)))
              nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
                (+
                  (f2cl-lib:fref a-%data%
                    (i j)

```

```

                                ((1 lda) (1 *))
                                a-%offset%)
    (*
      (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%)
      temp)))
    (setf ix (f2cl-lib:int-add ix incx))))
  (setf (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%)
    (coerce
      (+
        (coerce (realpart
                  (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%)) 'double-float)
        (coerce (realpart
                  (*
                    (f2cl-lib:fref x-%data%
                                    (jx)
                                    ((1 *))
                                    x-%offset%)
                    temp)) 'double-float))
      ' (complex double-float))))
  (t
    (setf (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%)
      (coerce
        (coerce (realpart
                  (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%)) 'double-float)
        ' (complex double-float))))
    (setf jx (f2cl-lib:int-add jx incx))))))
  (t
    (cond
      ((= incx 1)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j n) nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref x (j) ((1 *)) zero)
                (setf temp

```

```

(coerce
  (* alpha
    (f2cl-lib:dconjg
      (f2cl-lib:fref x-%data%
        (j)
        ((1 *))
        x-%offset%)))
    '(complex double-float)))
(setf (f2cl-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)

(coerce
  (+
    (coerce (realpart
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%)) 'double-float)
    (coerce (realpart
      (* temp
        (f2cl-lib:fref x-%data%
          (j)
          ((1 *))
          x-%offset%))) 'double-float))
    '(complex double-float)))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)

    (+
      (f2cl-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%)

      (*
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)

        temp))))))
(t
  (setf (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)

```



```

(coerce
  (coerce (realpart
    (f2cl-lib:fref a-%data%
      (j j)
      ((1 lda) (1 *))
      a-%offset%)) 'double-float)
  '(complex double-float))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
          (setf temp
            (coerce
              (* alpha
                (f2cl-lib:dconjg
                  (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)))
              '(complex double-float)))
          (setf (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%)
            (coerce
              (+
                (coerce (realpart
                  (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%)) 'double-float)
                (coerce (realpart
                  (* temp
                    (f2cl-lib:fref x-%data%
                      (jx)
                      ((1 *))
                      x-%offset%))) 'double-float))
              '(complex double-float)))
          (setf ix jx)
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf ix (f2cl-lib:int-add ix incx))
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))

```

```

                                a-%offset%)
(+
  (f2cl-lib:fref a-%data%
    (i j)
    ((1 lda) (1 *))
    a-%offset%)
  (*
    (f2cl-lib:fref x-%data%
      (ix)
      ((1 *))
      x-%offset%)
    temp))))))
(t
  (setf (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)
    (coerce
      (coerce (realpart
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)) 'double-float)
      '(complex double-float))))
  (setf jx (f2cl-lib:int-add jx incx))))))
end_label
(return (values nil nil nil nil nil nil nil))))

```

zhpmv BLAS

— zhpmv.input —

```

)set break resume
)sys rm -f zhpmv.output
)spool zhpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhpmv.help —

```
=====
zhpmv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZHPMV - perform the matrix-vector operation $y := \alpha A x + \beta y$,

SYNOPSIS

```
SUBROUTINE ZHPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y,
                  INCY )
```

```
COMPLEX*16    ALPHA, BETA
```

```
INTEGER       INCX, INCY, N
```

```
CHARACTER*1   UPLO
```

```
COMPLEX*16    AP( * ), X( * ), Y( * )
```

PURPOSE

ZHPMV performs the matrix-vector operation

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N

must be at least zero. Unchanged on exit.

- ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.
- AP - COMPLEX*16 array of DIMENSION at least
((n*(n + 1))/2). Before entry with UPLO = 'U'
or 'u', the array AP must contain the upper triangu-
lar part of the hermitian matrix packed sequentially,
column by column, so that AP(1) contains a(1, 1),
AP(2) and AP(3) contain a(1, 2) and a(2, 2)

respectively, and so on. Before entry with UPLO =
'L' or 'l', the array AP must contain the lower tri-
angular part of the hermitian matrix packed sequen-
tially, column by column, so that AP(1) contains a(
1, 1), AP(2) and AP(3) contain a(2, 1) and a(
3, 1) respectively, and so on. Note that the ima-
ginary parts of the diagonal elements need not be set
and are assumed to be zero. Unchanged on exit.
- X - COMPLEX*16 array of dimension at least
(1 + (n - 1)*abs(INCX)). Before entry, the
incremented array X must contain the n element vector
x. Unchanged on exit.
- INCX - INTEGER.
On entry, INCX specifies the increment for the ele-
ments of X. INCX must not be zero. Unchanged on
exit.
- BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA
is supplied as zero then Y need not be set on input.
Unchanged on exit.
- Y - COMPLEX*16 array of dimension at least
(1 + (n - 1)*abs(INCY)). Before entry, the
incremented array Y must contain the n element vector
y. On exit, Y is overwritten by the updated vector y.
- INCY - INTEGER.
On entry, INCY specifies the increment for the ele-
ments of Y. INCY must not be zero. Unchanged on
exit.
-

— zhpmv.f —

```

SUBROUTINE ZHPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
*
* .. Scalar Arguments ..
COMPLEX*16      ALPHA, BETA
INTEGER         INCX, INCY, N
CHARACTER*1     UPLO
*
* .. Array Arguments ..
COMPLEX*16      AP( * ), X( * ), Y( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16      ONE
PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16      ZERO
PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
COMPLEX*16      TEMP1, TEMP2
INTEGER         I, INFO, IX, IY, J, JX, JY, K, KK, KX, KY
*
* .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*
* .. External Subroutines ..
EXTERNAL        XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC       DCONJG, DBLE
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ) .AND.
$            .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = 1
      ELSE IF( N.LT.0 ) THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 6
      ELSE IF( INCY.EQ.0 ) THEN

```

```

        INFO = 9
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZHPMV ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   Set up the start points in X and Y.
*
      IF( INCX.GT.0 )THEN
        KX = 1
      ELSE
        KX = 1 - ( N - 1 )*INCX
      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( N - 1 )*INCY
      END IF
*
*   Start the operations. In this version the elements of the array AP
*   are accessed sequentially with one pass through AP.
*
*   First form y := beta*y.
*
      IF( BETA.NE.ONE )THEN
        IF( INCY.EQ.1 )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 10, I = 1, N
              Y( I ) = ZERO
10          CONTINUE
          ELSE
            DO 20, I = 1, N
              Y( I ) = BETA*Y( I )
20          CONTINUE
            END IF
          ELSE
            IY = KY
            IF( BETA.EQ.ZERO )THEN
              DO 30, I = 1, N
                Y( IY ) = ZERO
                IY = IY + INCY
30          CONTINUE
            ELSE
              DO 40, I = 1, N

```

```

        Y( IY ) = BETA*Y( IY )
        IY      = IY          + INCY
40      CONTINUE
      END IF
    END IF
  END IF
  IF( ALPHA.EQ.ZERO )
$    RETURN
    KK = 1
    IF( LSAME( UPLO, 'U' ) )THEN
*
*      Form y when AP contains the upper triangle.
*
    IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
      DO 60, J = 1, N
        TEMP1 = ALPHA*X( J )
        TEMP2 = ZERO
        K      = KK
        DO 50, I = 1, J - 1
          Y( I ) = Y( I ) + TEMP1*AP( K )
          TEMP2 = TEMP2 + DCONJG( AP( K ) ) * X( I )
          K      = K      + 1
50      CONTINUE
        Y( J ) = Y( J ) + TEMP1*DBLE( AP( KK + J - 1 ) )
$          + ALPHA*TEMP2
        KK      = KK      + J
60      CONTINUE
      ELSE
        JX = KX
        JY = KY
        DO 80, J = 1, N
          TEMP1 = ALPHA*X( JX )
          TEMP2 = ZERO
          IX    = KX
          IY    = KY
          DO 70, K = KK, KK + J - 2
            Y( IY ) = Y( IY ) + TEMP1*AP( K )
            TEMP2 = TEMP2 + DCONJG( AP( K ) ) * X( IX )
            IX    = IX    + INCX
            IY    = IY    + INCY
70      CONTINUE
          Y( JY ) = Y( JY ) + TEMP1*DBLE( AP( KK + J - 1 ) )
$            + ALPHA*TEMP2
          JX    = JX    + INCX
          JY    = JY    + INCY
          KK    = KK    + J
80      CONTINUE
        END IF
      ELSE
*

```

```

*      Form y when AP contains the lower triangle.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 100, J = 1, N
          TEMP1 = ALPHA*X( J )
          TEMP2 = ZERO
          Y( J ) = Y( J ) + TEMP1*DBLE( AP( KK ) )
          K      = KK      + 1
          DO 90, I = J + 1, N
            Y( I ) = Y( I ) + TEMP1*AP( K )
            TEMP2 = TEMP2 + DCONJG( AP( K ) ) * X( I )
            K      = K      + 1
          90      CONTINUE
          Y( J ) = Y( J ) + ALPHA*TEMP2
          KK      = KK      + ( N - J + 1 )
100      CONTINUE
        ELSE
          JX = KX
          JY = KY
          DO 120, J = 1, N
            TEMP1 = ALPHA*X( JX )
            TEMP2 = ZERO
            Y( JY ) = Y( JY ) + TEMP1*DBLE( AP( KK ) )
            IX      = JX
            IY      = JY
            DO 110, K = KK + 1, KK + N - J
              IX      = IX      + INCX
              IY      = IY      + INCY
              Y( IY ) = Y( IY ) + TEMP1*AP( K )
              TEMP2 = TEMP2 + DCONJG( AP( K ) ) * X( IX )
            110      CONTINUE
            Y( JY ) = Y( JY ) + ALPHA*TEMP2
            JX      = JX      + INCX
            JY      = JY      + INCY
            KK      = KK      + ( N - J + 1 )
120      CONTINUE
          END IF
        END IF
      *
      RETURN
      *
      *      End of ZHPMV .
      *
      END

```



```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
(declare (type (complex double-float) one) (type (complex double-float) zero))
(defun zhpmv (uplo n alpha ap x incx beta y incy)
  (declare (type (simple-array (complex double-float) (*)) y x ap)
    (type (complex double-float) beta alpha)
    (type fixnum incy incx n)
    (type character uplo))
  (f2cl-lib:with-multi-array-data
    ((uplo character uplo-%data% uplo-%offset%)
     (ap (complex double-float) ap-%data% ap-%offset%)
     (x (complex double-float) x-%data% x-%offset%)
     (y (complex double-float) y-%data% y-%offset%))
    (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
      (kx 0) (ky 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
      (declare (type fixnum i info ix iy j jx jy k kk kx ky)
        (type (complex double-float) temp1 temp2))
      (setf info 0)
      (cond
        ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
          (setf info 1))
        ((< n 0)
          (setf info 2))
        ((= incx 0)
          (setf info 6))
        ((= incy 0)
          (setf info 9)))
      (cond
        ((/= info 0)
          (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHPMV" info)
          (go end_label)))
      (if (or (= n 0) (and (= alpha zero) (= beta one))) (go end_label))
      (cond
        ((> incx 0)
          (setf kx 1))
        (t
          (setf kx
            (f2cl-lib:int-sub 1
              (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                incx)))))
      (cond
        ((> incy 0)
          (setf ky 1))
        (t
          (setf ky
            (f2cl-lib:int-sub 1
              (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                incy)))))
      (cond

```

```

( (/ = beta one)
  (cond
    ((= incy 1)
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i n) nil)

          (tagbody
            (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
              zero))))))
        (t
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i n) nil)

          (tagbody
            (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%
              (* beta
                (f2cl-lib:fref y-%data%
                  (i)
                  ((1 *))
                  y-%offset%))))))))))
      (t
        (setf iy ky)
        (cond
          ((= beta zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)

            (tagbody
              (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
                zero)
                (setf iy (f2cl-lib:int-add iy incy))))))
          (t
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i n) nil)

            (tagbody
              (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%
                (* beta
                  (f2cl-lib:fref y-%data%
                    (iy)
                    ((1 *))
                    y-%offset%)))
                (setf iy (f2cl-lib:int-add iy incy))))))))))
      (if (= alpha zero) (go end_label))
      (setf kk 1)
      (cond
        ((char-equal uplo #\U)
          (cond
            ((and (= incx 1) (= incy 1))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                (> j n) nil)

              (tagbody

```

```

(setf temp1
  (* alpha
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
(setf temp2 zero)
(setf k kk)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%))))))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:dconjg
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%))
      (f2cl-lib:fref x-%data%
        (i)
        ((1 *))
        x-%offset%))))))
(setf k (f2cl-lib:int-add k 1)))
(setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (* temp1
      (coerce (realpart
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk j)
            1))
          ((1 *))
          ap-%offset%)) 'double-float))
      (* alpha temp2)))
  (setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp1

```

```

(* alpha
  (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
(setf temp2 zero)
(setf ix kx)
(setf iy ky)
(f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add kk
      j
      (f2cl-lib:int-sub 2)))
    nil)
(tagbody
  (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
    (+
      (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
      (* temp1
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%))))))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:dconjg
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%))
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%))))))
(setf ix (f2cl-lib:int-add ix incx))
(setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
  (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
    (* temp1
      (coerce (realpart
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk j)
            1))
          ((1 *))
          ap-%offset%)) 'double-float))
      (* alpha temp2)))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (cond

```

```

((and (= incx 1) (= incy 1))
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
 (tagbody
  (setf temp1
   (* alpha
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)))
  (setf temp2 zero)
  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
   (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (* temp1
     (coerce (realpart
      (f2cl-lib:fref ap-%data%
       (kk)
       ((1 *))
       ap-%offset%)) 'double-float))))
  (setf k (f2cl-lib:int-add kk 1))
  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
   (f2cl-lib:int-add i 1))
   (> i n) nil)
  (tagbody
   (setf (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
    (+
     (f2cl-lib:fref y-%data% (i) ((1 *)) y-%offset%)
     (* temp1
      (f2cl-lib:fref ap-%data%
       (k)
       ((1 *))
       ap-%offset%))))
   (setf temp2
    (+ temp2
     (*
      (f2cl-lib:dconjg
       (f2cl-lib:fref ap-%data%
        (k)
        ((1 *))
        ap-%offset%))
      (f2cl-lib:fref x-%data%
       (i)
       ((1 *))
       x-%offset%))))
   (setf k (f2cl-lib:int-add k 1))))
  (setf (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
   (+ (f2cl-lib:fref y-%data% (j) ((1 *)) y-%offset%)
    (* alpha temp2)))
 (setf kk
  (f2cl-lib:int-add kk
   (f2cl-lib:int-add
    (f2cl-lib:int-sub n j)
    1))))))

```

```

(t
  (setf jx kx)
  (setf jy ky)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)))
    (setf temp2 zero)
    (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
      (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
        (* temp1
          (coerce (realpart
            (f2cl-lib:fref ap-%data%
              (kk)
              ((1 *))
              ap-%offset%)) 'double-float))))))
    (setf ix jx)
    (setf iy jy)
    (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
      (f2cl-lib:int-add k 1))
      (> k
        (f2cl-lib:int-add kk
          n
          (f2cl-lib:int-sub j)))
      nil)
    (tagbody
      (setf ix (f2cl-lib:int-add ix incx))
      (setf iy (f2cl-lib:int-add iy incy))
      (setf (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
        (+
          (f2cl-lib:fref y-%data% (iy) ((1 *)) y-%offset%)
          (* temp1
            (f2cl-lib:fref ap-%data%
              (k)
              ((1 *))
              ap-%offset%))))))
      (setf temp2
        (+ temp2
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%))
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%)))))))

```

```

      (setf (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
            (+ (f2cl-lib:fref y-%data% (jy) ((1 *)) y-%offset%)
               (* alpha temp2)))
      (setf jx (f2cl-lib:int-add jx incx))
      (setf jy (f2cl-lib:int-add jy incy))
      (setf kk
            (f2cl-lib:int-add kk
                              (f2cl-lib:int-add
                               (f2cl-lib:int-sub n j)
                               1))))))
end_label
  (return (values nil nil nil nil nil nil nil nil))))

```

zhpr2 BLAS

— zhpr2.input —

```

)set break resume
)sys rm -f zhpr2.output
)spool zhpr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhpr2.help —

```

=====
zhpr2 examples
=====

=====
Man Page Details
=====

```

NAME

ZHPR2 - perform the hermitian rank 2 operation $A := \alpha x x^* + \text{conjg}(\alpha) y y^* + A$,

SYNOPSIS

```
SUBROUTINE ZHPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
```

```
    COMPLEX*16    ALPHA
```

```
    INTEGER       INCX, INCY, N
```

```
    CHARACTER*1    UPLO
```

```
    COMPLEX*16    AP( * ), X( * ), Y( * )
```

PURPOSE

ZHPR2 performs the hermitian rank 2 operation

where alpha is a scalar, x and y are n element vectors and A is an n by n hermitian matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - COMPLEX*16 array of dimension at least (1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on

exit.

Y - COMPLEX*16 array of dimension at least
 (1 + (n - 1) * abs(INCY)). Before entry, the
 incremented array Y must contain the n element vector
 y. Unchanged on exit.

INCY - INTEGER.
 On entry, INCY specifies the increment for the ele-
 ments of Y. INCY must not be zero. Unchanged on
 exit.

AP - COMPLEX*16 array of DIMENSION at least
 ((n * (n + 1)) / 2). Before entry with UPLO = 'U'
 or 'u', the array AP must contain the upper triangu-
 lar part of the hermitian matrix packed sequentially,
 column by column, so that AP(1) contains a(1, 1),
 AP(2) and AP(3) contain a(1, 2) and a(2, 2)
 respectively, and so on. On exit, the array AP is
 overwritten by the upper triangular part of the
 updated matrix. Before entry with UPLO = 'L' or 'l',
 the array AP must contain the lower triangular part
 of the hermitian matrix packed sequentially, column
 by column, so that AP(1) contains a(1, 1), AP(2
) and AP(3) contain a(2, 1) and a(3, 1) respec-
 tively, and so on. On exit, the array AP is overwrit-
 ten by the lower triangular part of the updated
 matrix. Note that the imaginary parts of the diago-
 nal elements need not be set, they are assumed to be
 zero, and on exit they are set to zero.

— zhpr2.f —

```

SUBROUTINE ZHPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
* .. Scalar Arguments ..
  COMPLEX*16      ALPHA
  INTEGER         INCX, INCY, N
  CHARACTER*1     UPLO
* .. Array Arguments ..
  COMPLEX*16      AP( * ), X( * ), Y( * )
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.

```

```

*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
COMPLEX*16          ZERO
PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*      .. Local Scalars ..
COMPLEX*16          TEMP1, TEMP2
INTEGER             I, INFO, IX, IY, J, JX, JY, K, KK, KX, KY
*      .. External Functions ..
LOGICAL             LSAME
EXTERNAL            LSAME
*      .. External Subroutines ..
EXTERNAL            XERBLA
*      .. Intrinsic Functions ..
INTRINSIC           DCONJG, DBLE
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ) ).AND.
$           .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = 1
      ELSE IF( N.LT.0 ) THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 5
      ELSE IF( INCY.EQ.0 ) THEN
          INFO = 7
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZHPR2 ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.ZERO ) )
$      RETURN
*
*      Set up the start points in X and Y if the increments are not both
*      unity.
*
      IF( ( INCX.NE.1 ).OR.( INCY.NE.1 ) ) THEN
          IF( INCX.GT.0 ) THEN
              KX = 1
          ELSE
              KX = 1 - ( N - 1 ) * INCX
          
```

```

      END IF
      IF( INCY.GT.0 )THEN
        KY = 1
      ELSE
        KY = 1 - ( N - 1 )*INCY
      END IF
      JX = KX
      JY = KY
    END IF

*
*   Start the operations. In this version the elements of the array AP
*   are accessed sequentially with one pass through AP.
*
    KK = 1
    IF( LSAME( UPLO, 'U' ) )THEN

*
*   Form A when upper triangle is stored in AP.
*
      IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 20, J = 1, N
          IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
            TEMP1 = ALPHA*DCONJG( Y( J ) )
            TEMP2 = DCONJG( ALPHA*X( J ) )
            K      = KK
            DO 10, I = 1, J - 1
              AP( K ) = AP( K ) + X( I )*TEMP1 + Y( I )*TEMP2
              K        = K      + 1
10          CONTINUE
              AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) ) +
$              DBLE( X( J )*TEMP1 + Y( J )*TEMP2 )
              ELSE
                AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
              END IF
              KK = KK + J
20          CONTINUE
            ELSE
              DO 40, J = 1, N
                IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                  TEMP1 = ALPHA*DCONJG( Y( JY ) )
                  TEMP2 = DCONJG( ALPHA*X( JX ) )
                  IX     = KX
                  IY     = KY
                  DO 30, K = KK, KK + J - 2
                    AP( K ) = AP( K ) + X( IX )*TEMP1 + Y( IY )*TEMP2
                    IX      = IX      + INCX
                    IY      = IY      + INCY
30                  CONTINUE
                    AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) ) +
$                    DBLE( X( JX )*TEMP1 +
$                    Y( JY )*TEMP2 )

```

```

ELSE
    AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
END IF
JX = JX + INCX
JY = JY + INCY
KK = KK + J
40    CONTINUE
    END IF
ELSE
*
*    Form A when lower triangle is stored in AP.
*
    IF( ( INCX.EQ.1 ).AND.( INCY.EQ.1 ) )THEN
        DO 60, J = 1, N
            IF( ( X( J ).NE.ZERO ).OR.( Y( J ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*DCONJG( Y( J ) )
                TEMP2 = DCONJG( ALPHA*X( J ) )
                AP( KK ) = DBLE( AP( KK ) ) +
$                   DBLE( X( J )*TEMP1 + Y( J )*TEMP2 )
                K      = KK + 1
                DO 50, I = J + 1, N
                    AP( K ) = AP( K ) + X( I )*TEMP1 + Y( I )*TEMP2
                    K      = K + 1
50                CONTINUE
            ELSE
                AP( KK ) = DBLE( AP( KK ) )
            END IF
            KK = KK + N - J + 1
60        CONTINUE
    ELSE
        DO 80, J = 1, N
            IF( ( X( JX ).NE.ZERO ).OR.( Y( JY ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*DCONJG( Y( JY ) )
                TEMP2 = DCONJG( ALPHA*X( JX ) )
                AP( KK ) = DBLE( AP( KK ) ) +
$                   DBLE( X( JX )*TEMP1 + Y( JY )*TEMP2 )
                IX      = JX
                IY      = JY
                DO 70, K = KK + 1, KK + N - J
                    IX      = IX + INCX
                    IY      = IY + INCY
                    AP( K ) = AP( K ) + X( IX )*TEMP1 + Y( IY )*TEMP2
70                CONTINUE
            ELSE
                AP( KK ) = DBLE( AP( KK ) )
            END IF
            JX = JX + INCX
            JY = JY + INCY
            KK = KK + N - J + 1
80        CONTINUE
    
```

```

        END IF
    END IF
*
    RETURN
*
*   End of ZHPR2 .
*
END

```

— BLAS 2 zhpr2 —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zhpr2 (uplo n alpha x incx y incy ap)
    (declare (type (simple-array (complex double-float) (*)) ap y x)
              (type (complex double-float) alpha)
              (type fixnum incy incx n)
              (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (y (complex double-float) y-%data% y-%offset%)
       (ap (complex double-float) ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (iy 0) (j 0) (jx 0) (jy 0) (k 0) (kk 0)
              (kx 0) (ky 0) (temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)))
        (declare (type fixnum i info ix iy j jx jy k kk kx ky)
                  (type (complex double-float) temp1 temp2))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          ((< n 0)
            (setf info 2))
          ((= incx 0)
            (setf info 5))
          ((= incy 0)
            (setf info 7)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "ZHPR2" info)
            (go end_label)))
          (if (or (= n 0) (= alpha zero)) (go end_label))
        (cond
          ((or (/= incx 1) (/= incy 1))

```

```

(cond
  (> incx 0)
  (setf kx 1))
(t
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx))))))

(cond
  (> incy 0)
  (setf ky 1))
(t
  (setf ky
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incy))))))

(setf jx kx)
(setf jy ky))
(setf kk 1)
(cond
  ((char-equal uplo #\U)
   (cond
     ((and (= incx 1) (= incy 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond
          ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
                (/= (f2cl-lib:fref y (j) ((1 *))) zero))
           (setf temp1
             (* alpha
               (f2cl-lib:dconjg
                 (f2cl-lib:fref y-%data%
                   (j)
                   ((1 *))
                   y-%offset%))))))
          (setf temp2
            (coerce
              (f2cl-lib:dconjg
                (* alpha
                  (f2cl-lib:fref x-%data%
                    (j)
                    ((1 *))
                    x-%offset%)))
              '(complex double-float)))
           (setf k kk)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
             (> i

```

```

(f2cl-lib:int-add j
(f2cl-lib:int-sub 1)))
nil)
(tagbody
(setf (f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%)
(+
(f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%)
(*
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
temp1)
(*
(f2cl-lib:fref y-%data%
(i)
((1 *))
y-%offset%)
temp2)))
(setf k (f2cl-lib:int-add k 1)))
(setf (f2cl-lib:fref ap-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add kk j)
1))
((1 *))
ap-%offset%)
(coerce
(+
(coerce (realpart
(f2cl-lib:fref ap-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add kk j)
1))
((1 *))
ap-%offset%)) 'double-float)
(coerce (realpart
(+
(*
(f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)
temp1)
(*

```

```

(f2cl-lib:fref y-%data%
  (j)
  ((1 *))
  y-%offset%)
temp2))) 'double-float))
'(complex double-float)))
(t
  (setf (f2cl-lib:fref ap-%data%
    ((f2cl-lib:int-sub
      (f2cl-lib:int-add kk j)
      1))
    ((1 *))
    ap-%offset%)
    (coerce
      (coerce (realpart
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk j)
            1))
          ((1 *))
          ap-%offset%)) 'double-float)
        '(complex double-float))))))
  (setf kk (f2cl-lib:int-add kk j))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((or (/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (/= (f2cl-lib:fref y (jy) ((1 *))) zero))
        (setf temp1
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref y-%data%
                (jy)
                ((1 *))
                y-%offset%))))))
        (setf temp2
          (coerce
            (f2cl-lib:dconjg
              (* alpha
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%)))
            '(complex double-float)))
        (setf ix kx)
        (setf iy ky)
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
          (> k

```



```

                                (f2cl-lib:int-add kk
                                j
                                (f2cl-lib:int-sub 2)))
                                nil)
(tagbody
  (setf (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
    (+
      (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
      (*
        (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
        temp1)
      (*
        (f2cl-lib:fref y-%data%
                        (iy)
                        ((1 *))
                        y-%offset%)
        temp2)))
  (setf ix (f2cl-lib:int-add ix incx))
  (setf iy (f2cl-lib:int-add iy incy)))
(setf (f2cl-lib:fref ap-%data%
                    ((f2cl-lib:int-sub
                      (f2cl-lib:int-add kk j)
                      1))
                    ((1 *))
                    ap-%offset%)
  (coerce
    (+
      (coerce (realpart
        (f2cl-lib:fref ap-%data%
                      ((f2cl-lib:int-sub
                        (f2cl-lib:int-add kk j)
                        1))
                      ((1 *))
                      ap-%offset%)) 'double-float)
      (coerce (realpart
        (+
          (*
            (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)

```

```

temp1)
(*
  (f2cl-lib:fref y-%data%
                (jy)
                ((1 *))
                y-%offset%)
temp2))) 'double-float))
'(complex double-float)))
(t
  (setf (f2cl-lib:fref ap-%data%
                    ((f2cl-lib:int-sub
                     (f2cl-lib:int-add kk j)
                     1))
                    ((1 *))
                    ap-%offset%)
        (coerce
         (coerce (realpart
                   (f2cl-lib:fref ap-%data%
                                   ((f2cl-lib:int-sub
                                    (f2cl-lib:int-add kk j)
                                    1))
                                   ((1 *))
                                   ap-%offset%)) 'double-float)
         'double-float)))
  (setf jx (f2cl-lib:int-add jx incx))
  (setf jy (f2cl-lib:int-add jy incy))
  (setf kk (f2cl-lib:int-add kk j))))))
(t
  (cond
    ((and (= incx 1) (= incy 1))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (cond
        ((or (/= (f2cl-lib:fref x (j) ((1 *))) zero)
              (/= (f2cl-lib:fref y (j) ((1 *))) zero))
         (setf temp1
              (* alpha
                 (f2cl-lib:dconjg
                  (f2cl-lib:fref y-%data%
                                  (j)
                                  ((1 *))
                                  y-%offset%))))
         (setf temp2
              (coerce
               (f2cl-lib:dconjg
                (* alpha
                   (f2cl-lib:fref x-%data%
                                   (j)
                                   ((1 *))

```

```

                                x-%offset%)))
      '(complex double-float)))
(setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
(coerce
(+
(coerce (realpart
(f2cl-lib:fref ap-%data%
(kk)
((1 *))
ap-%offset%)) 'double-float)
(coerce (realpart
(+
(*
(f2cl-lib:fref x-%data%
(j)
((1 *))
x-%offset%)
temp1)
(*
(f2cl-lib:fref y-%data%
(j)
((1 *))
y-%offset%)
temp2))) 'double-float))
'(complex double-float)))
(setf k (f2cl-lib:int-add kk 1))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
(f2cl-lib:int-add i 1))
(> i n) nil)
(tagbody
(setf (f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%)
(+
(f2cl-lib:fref ap-%data%
(k)
((1 *))
ap-%offset%)
(*
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
temp1)
(*
(f2cl-lib:fref y-%data%
(i)
((1 *))
y-%offset%)

```

```

temp2)))
(setf k (f2cl-lib:int-add k 1))))))
(t
  (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
    (coerce
      (coerce (realpart
        (f2cl-lib:fref ap-%data%
          (kk)
            ((1 *))
              ap-%offset%)) 'double-float)
        '(complex double-float))))))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((or (/= (f2cl-lib:fref x (jx) ((1 *)) zero)
          (/= (f2cl-lib:fref y (jy) ((1 *)) zero))
          (setf temp1
            (* alpha
              (f2cl-lib:dconjg
                (f2cl-lib:fref y-%data%
                  (jy)
                    ((1 *))
                      y-%offset%))))))
          (setf temp2
            (coerce
              (f2cl-lib:dconjg
                (* alpha
                  (f2cl-lib:fref x-%data%
                    (jx)
                      ((1 *))
                        x-%offset%)))
                '(complex double-float)))
            (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
              (coerce
                (+
                  (coerce (realpart
                    (f2cl-lib:fref ap-%data%
                      (kk)
                        ((1 *))
                          ap-%offset%)) 'double-float)
                    (coerce (realpart
                      (+
                        (*
                          (f2cl-lib:fref x-%data%

```

```

                                (jx)
                                ((1 *))
                                x-%offset%)
    temp1)
  (*
    (f2cl-lib:fref y-%data%
                        (jy)
                        ((1 *))
                        y-%offset%)
    temp2))) 'double-float))
  '(complex double-float)))
(setf ix jx)
(setf iy jy)
(f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
                (f2cl-lib:int-add k 1))
              (> k
                (f2cl-lib:int-add kk
                                     n
                                     (f2cl-lib:int-sub j)))
              nil)
(tagbody
  (setf ix (f2cl-lib:int-add ix incx))
  (setf iy (f2cl-lib:int-add iy incy))
  (setf (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
        (+
          (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
          (*
            (f2cl-lib:fref x-%data%
                            (ix)
                            ((1 *))
                            x-%offset%)
            temp1)
          (*
            (f2cl-lib:fref y-%data%
                            (iy)
                            ((1 *))
                            y-%offset%)
            temp2))))))
(t
  (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
        (coerce
          (coerce (realpart
                    (f2cl-lib:fref ap-%data%
                                    (kk)

```

```

((1 *))
ap-%offset%) 'double-float)
'(complex double-float))))
(setf jx (f2cl-lib:int-add jx incx))
(setf jy (f2cl-lib:int-add jy incy))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

zhpr BLAS

— zhpr.input —

```

)set break resume
)sys rm -f zhpr.output
)spool zhpr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhpr.help —

```

=====
zhpr examples
=====

=====
Man Page Details
=====

NAME
  ZHPR - perform the hermitian rank 1 operation  A :=
  alpha*x*conj( x' ) + A,

SYNOPSIS

```

```
SUBROUTINE ZHPR ( UPLO, N, ALPHA, X, INCX, AP )
```

```
    DOUBLE      PRECISION ALPHA
```

```
    INTEGER      INCX, N
```

```
    CHARACTER*1  UPLO
```

```
    COMPLEX*16   AP( * ), X( * )
```

PURPOSE

ZHPR performs the hermitian rank 1 operation

where alpha is a real scalar, x is an n element vector and A is an n by n hermitian matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u' The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l' The lower triangular part of A is supplied in AP.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element vector x. Unchanged on exit.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

AP - COMPLEX*16 array of DIMENSION at least

((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. On exit, the array AP is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular part of the hermitian matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. On exit, the array AP is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

— zhpr.f —

```

SUBROUTINE ZHPR ( UPLO, N, ALPHA, X, INCX, AP )
*   .. Scalar Arguments ..
DOUBLE PRECISION  ALPHA
INTEGER           INCX, N
CHARACTER*1       UPLO
*   .. Array Arguments ..
COMPLEX*16        AP( * ), X( * )
*
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
*   .. Parameters ..
COMPLEX*16        ZERO
PARAMETER         ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*   .. Local Scalars ..
COMPLEX*16        TEMP
INTEGER           I, INFO, IX, J, JX, K, KK, KX
*   .. External Functions ..
LOGICAL           LSAME

```



```

      EXTERNAL          LSAME
*    .. External Subroutines ..
      EXTERNAL          XERBLA
*    .. Intrinsic Functions ..
      INTRINSIC          DCONJG, DBLE
*
*    .. Executable Statements ..
*
      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO, 'U' ).AND.
$          .NOT.LSAME( UPLO, 'L' )          )THEN
          INFO = 1
      ELSE IF( N.LT.0 )THEN
          INFO = 2
      ELSE IF( INCX.EQ.0 )THEN
          INFO = 5
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'ZHPR ', INFO )
          RETURN
      END IF
*
*    Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.( ALPHA.EQ.DBLE( ZERO ) ) )
$      RETURN
*
*    Set the start point in X if the increment is not unity.
*
      IF( INCX.LE.0 )THEN
          KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
          KX = 1
      END IF
*
*    Start the operations. In this version the elements of the array AP
*    are accessed sequentially with one pass through AP.
*
      KK = 1
      IF( LSAME( UPLO, 'U' ) )THEN
*
*        Form A when upper triangle is stored in AP.
*
          IF( INCX.EQ.1 )THEN
              DO 20, J = 1, N
                  IF( X( J ).NE.ZERO )THEN
                      TEMP = ALPHA*DCONJG( X( J ) )
                      K      = KK

```

```

DO 10, I = 1, J - 1
    AP( K ) = AP( K ) + X( I )*TEMP
    K      = K      + 1
10    CONTINUE
    AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
    $                + DBLE( X( J )*TEMP )
    ELSE
        AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
    END IF
    KK = KK + J
20    CONTINUE
    ELSE
        JX = KX
        DO 40, J = 1, N
            IF( X( JX ).NE.ZERO )THEN
                TEMP = ALPHA*DCONJG( X( JX ) )
                IX   = KX
                DO 30, K = KK, KK + J - 2
                    AP( K ) = AP( K ) + X( IX )*TEMP
                    IX      = IX      + INCX
30                CONTINUE
                    AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
                    $                + DBLE( X( JX )*TEMP )
                ELSE
                    AP( KK + J - 1 ) = DBLE( AP( KK + J - 1 ) )
                END IF
                JX = JX + INCX
                KK = KK + J
40            CONTINUE
        END IF
    ELSE
*
*      Form A when lower triangle is stored in AP.
*
        IF( INCX.EQ.1 )THEN
            DO 60, J = 1, N
                IF( X( J ).NE.ZERO )THEN
                    TEMP = ALPHA*DCONJG( X( J ) )
                    AP( KK ) = DBLE( AP( KK ) ) + DBLE( TEMP*X( J ) )
                    K      = KK      + 1
                    DO 50, I = J + 1, N
                        AP( K ) = AP( K ) + X( I )*TEMP
                        K      = K      + 1
50                    CONTINUE
                ELSE
                    AP( KK ) = DBLE( AP( KK ) )
                END IF
                KK = KK + N - J + 1
60            CONTINUE
        ELSE

```

```

      JX = KX
      DO 80, J = 1, N
        IF( X( JX ).NE.ZERO )THEN
          TEMP = ALPHA*DCONJG( X( JX ) )
          AP( KK ) = DBLE( AP( KK ) ) + DBLE( TEMP*X( JX ) )
          IX = JX
          DO 70, K = KK + 1, KK + N - J
            IX = IX + INCX
            AP( K ) = AP( K ) + X( IX )*TEMP
70          CONTINUE
          ELSE
            AP( KK ) = DBLE( AP( KK ) )
          END IF
          JX = JX + INCX
          KK = KK + N - J + 1
80        CONTINUE
      END IF
    END IF
*
*   RETURN
*
*   End of ZHPR .
*
*   END

```

— BLAS 2 zhpr —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun zhpr (uplo n alpha x incx ap)
    (declare (type (simple-array (complex double-float) (*)) ap x)
      (type (double-float) alpha)
      (type fixnum incx n)
      (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (x (complex double-float) x-%data% x-%offset%)
       (ap (complex double-float) ap-%data% ap-%offset%))
      (prog ((i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0) (kk 0) (kx 0)
        (temp #C(0.0 0.0)))
        (declare (type fixnum i info ix j jx k kk kx)
          (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))

```

```

(< n 0)
  (setf info 2))
  (= incx 0)
  (setf info 5)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZHPR" info)
  (go end_label)))
(if (or (= n 0) (= alpha (coerce (realpart zero) 'double-float)))
  (go end_label))
(cond
  (<= incx 0)
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx))))

  (/= incx 1)
  (setf kx 1)))
(setf kk 1)
(cond
  (char-equal uplo #\U)
  (cond
    (= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        (/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf temp
          (coerce
            (* alpha
              (f2cl-lib:dconjg
                (f2cl-lib:fref x-%data%
                  (j)
                  ((1 *))
                  x-%offset%)))
            '(complex double-float)))
        (setf k kk)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub 1)))
          nil)
        (tagbody
          (setf (f2cl-lib:fref ap-%data%
            (k)
            ((1 *))
            ap-%offset%

```

```

(+
  (f2cl-lib:fref ap-%data%
    (k)
    ((1 *))
    ap-%offset%)
  (*
    (f2cl-lib:fref x-%data%
      (i)
      ((1 *))
      x-%offset%)
    temp)))
(setf k (f2cl-lib:int-add k 1)))
(setf (f2cl-lib:fref ap-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add kk j)
    1))
  ((1 *))
  ap-%offset%))
(coerce
  (+
    (coerce (realpart
      (f2cl-lib:fref ap-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-add kk j)
          1))
        ((1 *))
        ap-%offset%)) 'double-float)
    (coerce (realpart
      (*
        (f2cl-lib:fref x-%data%
          (j)
          ((1 *))
          x-%offset%)
        temp)) 'double-float))
    'complex double-float)))
(t
  (setf (f2cl-lib:fref ap-%data%
    ((f2cl-lib:int-sub
      (f2cl-lib:int-add kk j)
      1))
    ((1 *))
    ap-%offset%))
  (coerce
    (coerce (realpart
      (f2cl-lib:fref ap-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-add kk j)
          1))
        ((1 *))
        ap-%offset%)) 'double-float)
    (coerce (realpart
      (*
        (f2cl-lib:fref x-%data%
          (j)
          ((1 *))
          x-%offset%)
        temp)) 'double-float))
    'complex double-float)))

```

```

                                '(complex double-float))))))
      (setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (coerce
              (* alpha
                (f2cl-lib:dconjg
                  (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)))
              '(complex double-float))))
          (setf ix kx)
          (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
            ((> k
              (f2cl-lib:int-add kk
                j
                (f2cl-lib:int-sub 2)))
              nil)
            (tagbody
              (setf (f2cl-lib:fref ap-%data%
                                (k)
                                ((1 *))
                                ap-%offset%)
                (+
                  (f2cl-lib:fref ap-%data%
                                (k)
                                ((1 *))
                                ap-%offset%)
                  (*
                    (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                    temp))))
              (setf ix (f2cl-lib:int-add ix incx))))
          (setf (f2cl-lib:fref ap-%data%
                                ((f2cl-lib:int-sub
                                  (f2cl-lib:int-add kk j)
                                  1))
                                ((1 *))
                                ap-%offset%)
            (coerce
              (+

```



```

(setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
      (coerce
        (+
          (coerce (realpart
                    (f2cl-lib:fref ap-%data%
                                   (kk)
                                   ((1 *))
                                   ap-%offset%)) 'double-float)
          (coerce (realpart
                    (* temp
                      (f2cl-lib:fref x-%data%
                                   (j)
                                   ((1 *))
                                   x-%offset%))) 'double-float))
          '(complex double-float)))
      (setf k (f2cl-lib:int-add kk 1))
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                    (f2cl-lib:int-add i 1))
                    (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref ap-%data%
                              (k)
                              ((1 *))
                              ap-%offset%)
              (+
                (f2cl-lib:fref ap-%data%
                              (k)
                              ((1 *))
                              ap-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                              (i)
                              ((1 *))
                              x-%offset%)
                  temp)))
        (setf k (f2cl-lib:int-add k 1))))
(t
 (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
       (coerce
         (coerce (realpart
                   (f2cl-lib:fref ap-%data%
                                   (kk)
                                   ((1 *))
                                   ap-%offset%)) 'double-float)
         '(complex double-float))))
(setf kk
  (f2cl-lib:int-add
   (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
   1))))
(t

```



```

(setf jx kx)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
      (setf temp
        (coerce
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)))
          '(complex double-float)))
      (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
        (coerce
          (+
            (coerce (realpart
              (f2cl-lib:fref ap-%data%
                (kk)
                ((1 *))
                ap-%offset%)) 'double-float)
            (coerce (realpart
              (* temp
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%))) 'double-float))
          '(complex double-float)))
      (setf ix jx)
      (f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
        (f2cl-lib:int-add k 1))
        (> k
          (f2cl-lib:int-add kk
            n
            (f2cl-lib:int-sub j)))
        nil)
      (tagbody
        (setf ix (f2cl-lib:int-add ix incx))
        (setf (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
          (+
            (f2cl-lib:fref ap-%data%
              (k)
              ((1 *))
              ap-%offset%)
            (*

```

```

                                (f2cl-lib:fref x-%data%
                                (ix)
                                ((1 *))
                                x-%offset%)
                                temp))))))
(t
  (setf (f2cl-lib:fref ap-%data% (kk) ((1 *)) ap-%offset%)
        (coerce
          (coerce (realpart
                    (f2cl-lib:fref ap-%data%
                                   (kk)
                                   ((1 *))
                                   ap-%offset%)) 'double-float)
          '(complex double-float))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add kk n) j)
    1))))))
end_label
(return (values nil nil nil nil nil nil))))

```

ztbmv BLAS

— ztbmv.input —

```

)set break resume
)sys rm -f ztbmv.output
)spool ztbmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztbmv.help —

```

=====
ztbmv examples
=====

```

```
=====
Man Page Details
=====
```

NAME

ZTBMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$, or $x := \text{conjg}(A')*x$,

SYNOPSIS

```
SUBROUTINE ZTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
                  )
```

```
INTEGER          INCX, K, LDA, N
```

```
CHARACTER*1      DIAG, TRANS, UPLO
```

```
COMPLEX*16       A( LDA, * ), X( * )
```

PURPOSE

ZTBMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := A*x$.

TRANS = 'T' or 't' $x := A'*x$.

TRANS = 'C' or 'c' $x := \text{conjg}(A')*x$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy 0 .le. K. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
K ), J A( M + I, J ) = matrix( I, J ) 10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
```

```

) A( M + I, J ) = matrix( I, J ) 10    CONTINUE 20
CONTINUE

```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.

X - COMPLEX*16 array of dimension at least
(1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— ztbmv.f —

```

SUBROUTINE ZTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
*
* .. Scalar Arguments ..
INTEGER          INCX, K, LDA, N
CHARACTER*1      DIAG, TRANS, UPLO
*
* .. Array Arguments ..
COMPLEX*16       A( LDA, * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16       ZERO
PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )

```

```

*    .. Local Scalars ..
COMPLEX*16      TEMP
INTEGER         I, INFO, IX, J, JX, KPLUS1, KX, L
LOGICAL         NOCONJ, NOUNIT
*    .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*    .. External Subroutines ..
EXTERNAL        XERBLA
*    .. Intrinsic Functions ..
INTRINSIC       DCONJG, MAX, MIN
*    ..
*    .. Executable Statements ..
*
*    Test the input parameters.
*
    INFO = 0
    IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$          .NOT.LSAME( UPLO , 'L' )      )THEN
        INFO = 1
    ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$          .NOT.LSAME( TRANS, 'T' ).AND.
$          .NOT.LSAME( TRANS, 'C' )      )THEN
        INFO = 2
    ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$          .NOT.LSAME( DIAG , 'N' )      )THEN
        INFO = 3
    ELSE IF( N.LT.0 )THEN
        INFO = 4
    ELSE IF( K.LT.0 )THEN
        INFO = 5
    ELSE IF( LDA.LT.( K + 1 ) )THEN
        INFO = 7
    ELSE IF( INCX.EQ.0 )THEN
        INFO = 9
    END IF
    IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZTBMV ', INFO )
        RETURN
    END IF
*
*    Quick return if possible.
*
    IF( N.EQ.0 )
$      RETURN
*
    NOCONJ = LSAME( TRANS, 'T' )
    NOUNIT = LSAME( DIAG , 'N' )
*
*    Set up the start point in X if the increment is not unity. This

```

```

*      will be ( N - 1 ) * INCX   too small for descending loops.
*
      IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 ) * INCX
      ELSE IF( INCX.NE.1 )THEN
        KX = 1
      END IF
*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through A.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*        Form x := A*x.
*
        IF( LSAME( UPLO, 'U' ) )THEN
          KPLUS1 = K + 1
          IF( INCX.EQ.1 )THEN
            DO 20, J = 1, N
              IF( X( J ).NE.ZERO )THEN
                TEMP = X( J )
                L = KPLUS1 - J
                DO 10, I = MAX( 1, J - K ), J - 1
                  X( I ) = X( I ) + TEMP * A( L + I, J )
10              CONTINUE
              IF( NOUNIT )
                $          X( J ) = X( J ) * A( KPLUS1, J )
              END IF
20            CONTINUE
          ELSE
            JX = KX
            DO 40, J = 1, N
              IF( X( JX ).NE.ZERO )THEN
                TEMP = X( JX )
                IX = KX
                L = KPLUS1 - J
                DO 30, I = MAX( 1, J - K ), J - 1
                  X( IX ) = X( IX ) + TEMP * A( L + I, J )
                  IX = IX + INCX
30              CONTINUE
              IF( NOUNIT )
                $          X( JX ) = X( JX ) * A( KPLUS1, J )
              END IF
              JX = JX + INCX
              IF( J.GT.K )
                $          KX = KX + INCX
40            CONTINUE
          END IF
        ELSE
          IF( INCX.EQ.1 )THEN

```

```

DO 60, J = N, 1, -1
  IF( X( J ).NE.ZERO )THEN
    TEMP = X( J )
    L = 1 - J
    DO 50, I = MIN( N, J + K ), J + 1, -1
      X( I ) = X( I ) + TEMP*A( L + I, J )
50    CONTINUE
    IF( NOUNIT )
      $      X( J ) = X( J )*A( 1, J )
    END IF
60    CONTINUE
  ELSE
    KX = KX + ( N - 1 )*INCX
    JX = KX
    DO 80, J = N, 1, -1
      IF( X( JX ).NE.ZERO )THEN
        TEMP = X( JX )
        IX = KX
        L = 1 - J
        DO 70, I = MIN( N, J + K ), J + 1, -1
          X( IX ) = X( IX ) + TEMP*A( L + I, J )
          IX = IX - INCX
70        CONTINUE
        IF( NOUNIT )
          $      X( JX ) = X( JX )*A( 1, J )
        END IF
        JX = JX - INCX
        IF( ( N - J ).GE.K )
          $      KX = KX - INCX
80        CONTINUE
      END IF
    END IF
  ELSE
    *
    *      Form x := A'*x or x := conjg( A' )*x.
    *
    IF( LSAME( UPLO, 'U' ) )THEN
      KPLUS1 = K + 1
      IF( INCX.EQ.1 )THEN
        DO 110, J = N, 1, -1
          TEMP = X( J )
          L = KPLUS1 - J
          IF( NOCONJ )THEN
            IF( NOUNIT )
              $      TEMP = TEMP*A( KPLUS1, J )
            DO 90, I = J - 1, MAX( 1, J - K ), -1
              TEMP = TEMP + A( L + I, J )*X( I )
90            CONTINUE
          ELSE
            IF( NOUNIT )

```



```

$          TEMP = TEMP*DCONJG( A( KPLUS1, J ) )
          DO 100, I = J - 1, MAX( 1, J - K ), -1
            TEMP = TEMP + DCONJG( A( L + I, J ) ) * X( I )
100      CONTINUE
          END IF
          X( J ) = TEMP
110      CONTINUE
        ELSE
          KX = KX + ( N - 1 ) * INCX
          JX = KX
          DO 140, J = N, 1, -1
            TEMP = X( JX )
            KX = KX - INCX
            IX = KX
            L = KPLUS1 - J
            IF( NOCONJ ) THEN
              IF( NUNIT )
                $          TEMP = TEMP * A( KPLUS1, J )
                DO 120, I = J - 1, MAX( 1, J - K ), -1
                  TEMP = TEMP + A( L + I, J ) * X( IX )
                  IX = IX - INCX
120          CONTINUE
              ELSE
                IF( NUNIT )
                  $          TEMP = TEMP * DCONJG( A( KPLUS1, J ) )
                  DO 130, I = J - 1, MAX( 1, J - K ), -1
                    TEMP = TEMP + DCONJG( A( L + I, J ) ) * X( IX )
                    IX = IX - INCX
130          CONTINUE
                  END IF
                  X( JX ) = TEMP
                  JX = JX - INCX
140          CONTINUE
              END IF
            ELSE
              IF( INCX.EQ.1 ) THEN
                DO 170, J = 1, N
                  TEMP = X( J )
                  L = 1 - J
                  IF( NOCONJ ) THEN
                    IF( NUNIT )
                      $          TEMP = TEMP * A( 1, J )
                      DO 150, I = J + 1, MIN( N, J + K )
                        TEMP = TEMP + A( L + I, J ) * X( I )
150          CONTINUE
                    ELSE
                      IF( NUNIT )
                        $          TEMP = TEMP * DCONJG( A( 1, J ) )
                        DO 160, I = J + 1, MIN( N, J + K )
                          TEMP = TEMP + DCONJG( A( L + I, J ) ) * X( I )

```

```

160          CONTINUE
          END IF
          X( J ) = TEMP
170      CONTINUE
      ELSE
          JX = KX
          DO 200, J = 1, N
              TEMP = X( JX )
              KX = KX + INCX
              IX = KX
              L = 1 - J
              IF( NOCONJ ) THEN
                  IF( NOUNIT )
                      $          TEMP = TEMP*A( 1, J )
                      DO 180, I = J + 1, MIN( N, J + K )
                          TEMP = TEMP + A( L + I, J ) * X( IX )
                          IX = IX + INCX
180                  CONTINUE
                      ELSE
                          IF( NOUNIT )
                              $          TEMP = TEMP*DCONJG( A( 1, J ) )
                              DO 190, I = J + 1, MIN( N, J + K )
                                  TEMP = TEMP + DCONJG( A( L + I, J ) ) * X( IX )
                                  IX = IX + INCX
190                  CONTINUE
                      END IF
                      X( JX ) = TEMP
                      JX = JX + INCX
200          CONTINUE
          END IF
      END IF
  END IF
*
  RETURN
*
*   End of ZTBMV .
*
  END

```

— BLAS 2 ztbmv —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztbmv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
      (type fixnum incx lda k n)

```

```

        (type character diag trans uplo))
(f2c1-lib:with-multi-array-data
  ((uplo character uplo-%data% uplo-%offset%)
   (trans character trans-%data% trans-%offset%)
   (diag character diag-%data% diag-%offset%)
   (a (complex double-float) a-%data% a-%offset%)
   (x (complex double-float) x-%data% x-%offset%))
(prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
      (kplus1 0) (kx 0) (l 0) (temp #C(0.0 0.0)))
(declare (type (member t nil) noconj nounit)
          (type fixnum i info ix j jx kplus1 kx l)
          (type (complex double-float) temp))
(setf info 0)
(cond
  ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
   (setf info 1))
  ((and (not (char-equal trans #\N))
        (not (char-equal trans #\T))
        (not (char-equal trans #\C)))
   (setf info 2))
  ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
   (setf info 3))
  ((< n 0)
   (setf info 4))
  ((< k 0)
   (setf info 5))
  ((< lda (f2c1-lib:int-add k 1))
   (setf info 7))
  ((= incx 0)
   (setf info 9)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZTBMV" info)
   (go end_label)))
(if (= n 0) (go end_label))
(setf noconj (char-equal trans #\T))
(setf nounit (char-equal diag #\N))
(cond
  ((<= incx 0)
   (setf kx
    (f2c1-lib:int-sub 1
     (f2c1-lib:int-mul (f2c1-lib:int-sub n 1)
                       incx))))
  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond

```

```

(char-equal uplo #\U)
(setf kplus1 (f2cl-lib:int-add k 1))
(cond
  ((= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
          (setf l (f2cl-lib:int-sub kplus1 j))
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))
              (f2cl-lib:int-add i 1))
            (> i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub
                  1))))
            nil)
          (tagbody
            (setf (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%)
              (+
                (f2cl-lib:fref x-%data%
                  (i)
                  ((1 *))
                  x-%offset%)
                (* temp
                  (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add 1 i)
                     j)
                    ((1 lda) (1 *))
                    a-%offset%))))))
          (if nounit
            (setf (f2cl-lib:fref x-%data%
              (j)
              ((1 *))
              x-%offset%)
              (*
                (f2cl-lib:fref x-%data%
                  (j)
                  ((1 *))
                  x-%offset%)

```

```

                                (f2cl-lib:fref a-%data%
                                (kplus1 j)
                                ((1 lda) (1 *))
                                a-%offset%)))))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf temp
          (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%))
        (setf ix kx)
        (setf l (f2cl-lib:int-sub kplus1 j))
        (f2cl-lib:fdo (i
          (max (the fixnum 1)
            (the fixnum
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub
                  k))))
            (f2cl-lib:int-add i 1))
          (> i
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub
                1)))
            nil)
          (tagbody
            (setf (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))
              (+
                (f2cl-lib:fref x-%data%
                  (ix)
                  ((1 *))
                  x-%offset%)
                (* temp
                  (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add 1 i)
                      j)
                    ((1 lda) (1 *))
                    a-%offset%))))
              (setf ix (f2cl-lib:int-add ix incx))))
        (if nunit
          (setf (f2cl-lib:fref x-%data%
            (jx)

```

```

((1 *))
x-%offset%)
(*
(f2cl-lib:fref x-%data%
(jx)
((1 *))
x-%offset%)
(f2cl-lib:fref a-%data%
(kplus1 j)
((1 lda) (1 *))
a-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))
(if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))
(t
(cond
(= incx 1)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
(> j 1) nil)
(tagbody
(cond
(/= (f2cl-lib:fref x (j) ((1 *))) zero)
(setf temp
(f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(setf l (f2cl-lib:int-sub 1 j))
(f2cl-lib:fdo (i
(min (the fixnum n)
(the fixnum
(f2cl-lib:int-add j k)))
(f2cl-lib:int-add i
(f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1)) nil)
(tagbody
(setf (f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
(+
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)
(* temp
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add 1 i)
j)
((1 lda) (1 *))
a-%offset%))))))
(if nounit
(setf (f2cl-lib:fref x-%data%
(j)

```

```

                                ((1 *))
                                x-%offset%)
(*
  (f2cl-lib:fref x-%data%
    (j)
    ((1 *))
    x-%offset%)
  (f2cl-lib:fref a-%data%
    (1 j)
    ((1 lda) (1 *))
    a-%offset%)))))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j 1) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%))
          (setf ix kx)
          (setf l (f2cl-lib:int-sub 1 j))
          (f2cl-lib:fdo (i
            (min (the fixnum n)
              (the fixnum
                (f2cl-lib:int-add j k)))
            (f2cl-lib:int-add i
              (f2cl-lib:int-sub 1)))
            ((> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
                (+
                  (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%)
                  (* temp
                    (f2cl-lib:fref a-%data%
                      ((f2cl-lib:int-add 1 i)

```

```

                                j)
                                ((1 lda) (1 *))
                                a-%offset%)))
      (setf ix (f2cl-lib:int-sub ix incx))))
    (if nounit
      (setf (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
              (f2cl-lib:fref a-%data%
                          (1 j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
      (setf jx (f2cl-lib:int-sub jx incx))
      (if (>= (f2cl-lib:int-sub n j) k)
        (setf kx (f2cl-lib:int-sub kx incx)))))))))
  (t
   (cond
    ((char-equal uplo #\U)
     (setf kplus1 (f2cl-lib:int-add k 1))
     (cond
      ((= incx 1)
       (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                     (> j 1) nil)
       (tagbody
        (setf temp
                  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf l (f2cl-lib:int-sub kplus1 j))
        (cond
         (noconj
          (if nounit
              (setf temp
                    (* temp
                      (f2cl-lib:fref a-%data%
                                      (kplus1 j)
                                      ((1 lda) (1 *))
                                      a-%offset%))))
          (f2cl-lib:fdo (i
                        (f2cl-lib:int-add j
                          (f2cl-lib:int-sub 1))
                        (f2cl-lib:int-add i
                          (f2cl-lib:int-sub 1)))
                      (> i
                        (max (the fixnum 1)
                             (the fixnum

```



```

                                (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                 k))))))
                                nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add 1 i)
                         j)
                        ((1 lda) (1 *))
                        a-%offset%)
        (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%))))))
(t
  (if nount
    (setf temp
      (* temp
        (f2cl-lib:dconjg
         (f2cl-lib:fref a-%data%
                        (kplus1 j)
                        ((1 lda) (1 *))
                        a-%offset%))))))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
                    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
                    (f2cl-lib:int-sub 1))
  (> i
    (max (the fixnum 1)
          (the fixnum
            (f2cl-lib:int-add j
                              (f2cl-lib:int-sub
                               k))))))
    nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:dconjg
         (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add 1 i)
                         j)
                        ((1 lda) (1 *))
                        a-%offset%)
        (f2cl-lib:fref x-%data%
                        (i)

```

```

((1 *))
x-%offset%))))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
temp)))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (setf jx kx)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf kx (f2cl-lib:int-sub kx incx))
      (setf ix kx)
      (setf l (f2cl-lib:int-sub kplus1 j))
      (cond
        (noconj
          (if nounit
            (setf temp
              (* temp
                (f2cl-lib:fref a-%data%
                  (kplus1 j)
                  ((1 lda) (1 *))
                  a-%offset%))))
            (f2cl-lib:fdo (i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub 1))
              (f2cl-lib:int-add i
                (f2cl-lib:int-sub 1)))
              (> i
                (max (the fixnum 1)
                  (the fixnum
                    (f2cl-lib:int-add j
                      (f2cl-lib:int-sub
                        k))))))
                nil)
              (tagbody
                (setf temp
                  (+ temp
                    (*
                      (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add l i)
                          j)
                        ((1 lda) (1 *))
                        a-%offset%)
                      (f2cl-lib:fref x-%data%

```

```

                                (ix)
                                ((1 *))
                                x-%offset%))))
      (setf ix (f2cl-lib:int-sub ix incx))))))
(t
  (if nunit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
                        (kplus1 j)
                        ((1 lda) (1 *))
                        a-%offset%))))))
    (f2cl-lib:fdo (i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub 1))
      (f2cl-lib:int-add i
        (f2cl-lib:int-sub 1)))
      (> i
        (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))))
      nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                            ((f2cl-lib:int-add 1 i)
                             j)
                            ((1 lda) (1 *))
                            a-%offset%))
            (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%))))
          (setf ix (f2cl-lib:int-sub ix incx))))))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
        temp)
      (setf jx (f2cl-lib:int-sub jx incx))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp

```

```

(f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(setf l (f2cl-lib:int-sub 1 j))
(cond
  (noconj
    (if nount
      (setf temp
        (* temp
          (f2cl-lib:fref a-%data%
            (1 j)
            ((1 lda) (1 *))
            a-%offset%))))
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add i 1))
        (> i
          (min (the fixnum n)
            (the fixnum
              (f2cl-lib:int-add j k))))
          nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:fref a-%data%
                ((f2cl-lib:int-add 1 i)
                j)
                ((1 lda) (1 *))
                a-%offset%)
              (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
                x-%offset%)))))))
    (t
      (if nount
        (setf temp
          (* temp
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (1 j)
                ((1 lda) (1 *))
                a-%offset%))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
            (> i
              (min (the fixnum n)
                (the fixnum
                  (f2cl-lib:int-add j k))))
              nil)
          (tagbody
            (setf temp
              (+ temp

```

```

(*
(f2cl-lib:dconjg
  (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-add 1 i)
     j)
    ((1 lda) (1 *))
    a-%offset%))
  (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%))))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp)))
(t
 (setf jx kx)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
 (tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
  (setf kx (f2cl-lib:int-add kx incx))
  (setf ix kx)
  (setf l (f2cl-lib:int-sub 1 j))
  (cond
   (noconj
    (if nunit
     (setf temp
      (* temp
        (f2cl-lib:fref a-%data%
          (1 j)
          ((1 lda) (1 *))
          a-%offset%))))
     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      (> i
       (min (the fixnum n)
            (the fixnum
              (f2cl-lib:int-add j k))))
      nil)
     (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:fref a-%data%
              ((f2cl-lib:int-add 1 i)
               j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref x-%data%
              (ix)

```

```

((1 *))
x-%offset%))))
(setf ix (f2cl-lib:int-add ix incx))))))
(t
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (1 j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      (> i
        (min (the fixnum n)
          (the fixnum
            (f2cl-lib:int-add j k))))
      nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                ((f2cl-lib:int-add 1 i)
                  j)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))
          (setf ix (f2cl-lib:int-add ix incx))))))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
        temp)
      (setf jx (f2cl-lib:int-add jx incx)))))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))))

```

ztbsv BLAS

— ztbsv.input —

```

)set break resume
)sys rm -f ztbsv.output
)spool ztbsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztbsv.help —

```

=====
ztbsv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZTBSV - solve one of the systems of equations $Ax = b$, or $A'^*x = b$, or $\text{conjg}(A')^*x = b$,

SYNOPSIS

```

SUBROUTINE ZTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX
)

```

INTEGER INCX, K, LDA, N

CHARACTER*1 DIAG, TRANS, UPLO

COMPLEX*16 A(LDA, *), X(*)

PURPOSE

ZTBSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular band matrix, with (k + 1) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $A*x = b$.

TRANS = 'T' or 't' $A'*x = b$.

TRANS = 'C' or 'c' $\text{conjg}(A)*x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with UPLO = 'U' or 'u', K specifies the number of super-diagonals of the matrix A. On entry with UPLO = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A. K must satisfy $0 \leq K$. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading (k + 1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k + 1) of the array, the

first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = K + 1 - J DO 10, I = MAX( 1, J -
K ), J A( M + I, J ) = matrix( I, J ) 10    CONTINUE
20 CONTINUE
```

Before entry with UPLO = 'L' or 'l', the leading (k + 1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N M = 1 - J DO 10, I = J, MIN( N, J + K
) A( M + I, J ) = matrix( I, J ) 10    CONTINUE 20
CONTINUE
```

Note that when DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the

matrix are not referenced, but are assumed to be unity. Unchanged on exit.

- LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least (k + 1). Unchanged on exit.
- X - COMPLEX*16 array of dimension at least
(1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
- INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.
-

— ztbsv.f —

```

SUBROUTINE ZTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
*
* .. Scalar Arguments ..
INTEGER          INCX, K, LDA, N
CHARACTER*1      DIAG, TRANS, UPLO
*
* .. Array Arguments ..
COMPLEX*16       A( LDA, * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
COMPLEX*16       ZERO
PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
COMPLEX*16       TEMP
INTEGER          I, INFO, IX, J, JX, KPLUS1, KX, L
LOGICAL          NOCONJ, NOUNIT
*
* .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
*
* .. External Subroutines ..
EXTERNAL         XERBLA
*
* .. Intrinsic Functions ..
INTRINSIC        DCONJG, MAX, MIN
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
INFO = 0
IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$        .NOT.LSAME( UPLO , 'L' )      )THEN
    INFO = 1
ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$        .NOT.LSAME( TRANS, 'T' ).AND.
$        .NOT.LSAME( TRANS, 'C' )      )THEN
    INFO = 2
ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$        .NOT.LSAME( DIAG , 'N' )      )THEN
    INFO = 3

```

```

ELSE IF( N.LT.0 )THEN
    INFO = 4
ELSE IF( K.LT.0 )THEN
    INFO = 5
ELSE IF( LDA.LT.( K + 1 ) )THEN
    INFO = 7
ELSE IF( INCX.EQ.0 )THEN
    INFO = 9
END IF
IF( INFO.NE.0 )THEN
    CALL XERBLA( 'ZTBSV ', INFO )
    RETURN
END IF

*
*   Quick return if possible.
*
    IF( N.EQ.0 )
$   RETURN
*

NOCONJ = LSAME( TRANS, 'T' )
NOUNIT = LSAME( DIAG, 'N' )
*
*   Set up the start point in X if the increment is not unity. This
*   will be ( N - 1 )*INCX too small for descending loops.
*
    IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 )*INCX
    ELSE IF( INCX.NE.1 )THEN
        KX = 1
    END IF
*
*   Start the operations. In this version the elements of A are
*   accessed by sequentially with one pass through A.
*
    IF( LSAME( TRANS, 'N' ) )THEN
*
*       Form x := inv( A )*x.
*
        IF( LSAME( UPLO, 'U' ) )THEN
            KPLUS1 = K + 1
            IF( INCX.EQ.1 )THEN
                DO 20, J = N, 1, -1
                    IF( X( J ).NE.ZERO )THEN
                        L = KPLUS1 - J
                        IF( NOUNIT )
$                           X( J ) = X( J )/A( KPLUS1, J )
                        TEMP = X( J )
                        DO 10, I = J - 1, MAX( 1, J - K ), -1
                            X( I ) = X( I ) - TEMP*A( L + I, J )
10                        CONTINUE

```

```

                END IF
20      CONTINUE
      ELSE
        KX = KX + ( N - 1 ) * INCX
        JX = KX
        DO 40, J = N, 1, -1
          KX = KX - INCX
          IF( X( JX ).NE.ZERO ) THEN
            IX = KX
            L = KPLUS1 - J
            IF( NOUNIT )
$              X( JX ) = X( JX ) / A( KPLUS1, J )
            TEMP = X( JX )
            DO 30, I = J - 1, MAX( 1, J - K ), -1
              X( IX ) = X( IX ) - TEMP * A( L + I, J )
              IX = IX - INCX
30          CONTINUE
            END IF
            JX = JX - INCX
40      CONTINUE
      END IF
    ELSE
      IF( INCX.EQ.1 ) THEN
        DO 60, J = 1, N
          IF( X( J ).NE.ZERO ) THEN
            L = 1 - J
            IF( NOUNIT )
$              X( J ) = X( J ) / A( 1, J )
            TEMP = X( J )
            DO 50, I = J + 1, MIN( N, J + K )
              X( I ) = X( I ) - TEMP * A( L + I, J )
50          CONTINUE
            END IF
60      CONTINUE
      ELSE
        JX = KX
        DO 80, J = 1, N
          KX = KX + INCX
          IF( X( JX ).NE.ZERO ) THEN
            IX = KX
            L = 1 - J
            IF( NOUNIT )
$              X( JX ) = X( JX ) / A( 1, J )
            TEMP = X( JX )
            DO 70, I = J + 1, MIN( N, J + K )
              X( IX ) = X( IX ) - TEMP * A( L + I, J )
              IX = IX + INCX
70          CONTINUE
            END IF
            JX = JX + INCX

```

```

80          CONTINUE
          END IF
        END IF
      ELSE
*
*        Form x := inv( A' ) * x or x := inv( conjg( A' ) ) * x.
*
        IF( LSAME( UPLO, 'U' ) ) THEN
          KPLUS1 = K + 1
          IF( INCX.EQ.1 ) THEN
            DO 110, J = 1, N
              TEMP = X( J )
              L     = KPLUS1 - J
              IF( NOCONJ ) THEN
                DO 90, I = MAX( 1, J - K ), J - 1
                  TEMP = TEMP - A( L + I, J ) * X( I )
90              CONTINUE
                IF( NOUNIT )
                  $      TEMP = TEMP / A( KPLUS1, J )
              ELSE
                DO 100, I = MAX( 1, J - K ), J - 1
                  TEMP = TEMP - DCONJG( A( L + I, J ) ) * X( I )
100             CONTINUE
                IF( NOUNIT )
                  $      TEMP = TEMP / DCONJG( A( KPLUS1, J ) )
              END IF
              X( J ) = TEMP
110          CONTINUE
        ELSE
          JX = KX
          DO 140, J = 1, N
            TEMP = X( JX )
            IX   = KX
            L     = KPLUS1 - J
            IF( NOCONJ ) THEN
              DO 120, I = MAX( 1, J - K ), J - 1
                TEMP = TEMP - A( L + I, J ) * X( IX )
                IX   = IX + INCX
120             CONTINUE
                IF( NOUNIT )
                  $      TEMP = TEMP / A( KPLUS1, J )
              ELSE
                DO 130, I = MAX( 1, J - K ), J - 1
                  TEMP = TEMP - DCONJG( A( L + I, J ) ) * X( IX )
                  IX   = IX + INCX
130             CONTINUE
                IF( NOUNIT )
                  $      TEMP = TEMP / DCONJG( A( KPLUS1, J ) )
              END IF
            X( JX ) = TEMP

```

```

      JX      = JX  + INCX
      IF( J.GT.K )
$         KX = KX + INCX
140      CONTINUE
      END IF
      ELSE
      IF( INCX.EQ.1 )THEN
      DO 170, J = N, 1, -1
      TEMP = X( J )
      L    = 1      - J
      IF( NOCONJ )THEN
      DO 150, I = MIN( N, J + K ), J + 1, -1
      TEMP = TEMP - A( L + I, J ) * X( I )
150      CONTINUE
      IF( NOUNIT )
$         TEMP = TEMP / A( 1, J )
      ELSE
      DO 160, I = MIN( N, J + K ), J + 1, -1
      TEMP = TEMP - DCONJG( A( L + I, J ) ) * X( I )
160      CONTINUE
      IF( NOUNIT )
$         TEMP = TEMP / DCONJG( A( 1, J ) )
      END IF
      X( J ) = TEMP
170      CONTINUE
      ELSE
      KX = KX + ( N - 1 ) * INCX
      JX = KX
      DO 200, J = N, 1, -1
      TEMP = X( JX )
      IX  = KX
      L    = 1      - J
      IF( NOCONJ )THEN
      DO 180, I = MIN( N, J + K ), J + 1, -1
      TEMP = TEMP - A( L + I, J ) * X( IX )
      IX  = IX  - INCX
180      CONTINUE
      IF( NOUNIT )
$         TEMP = TEMP / A( 1, J )
      ELSE
      DO 190, I = MIN( N, J + K ), J + 1, -1
      TEMP = TEMP - DCONJG( A( L + I, J ) ) * X( IX )
      IX  = IX  - INCX
190      CONTINUE
      IF( NOUNIT )
$         TEMP = TEMP / DCONJG( A( 1, J ) )
      END IF
      X( JX ) = TEMP
      JX      = JX  - INCX
      IF( ( N - J ).GE.K )

```

```

$          KX = KX - INCX
200        CONTINUE
          END IF
        END IF
      END IF
*
      RETURN
*
*   End of ZTBSV .
*
      END

```

— BLAS 2 ztbsv —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztbsv (uplo trans diag n k a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
      (type fixnum incx lda k n)
      (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kplus1 0) (kx 0) (l 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
          (type fixnum i info ix j jx kplus1 kx l)
          (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
            (setf info 2))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
            (setf info 3))
          ((< n 0)
            (setf info 4))
          ((< k 0)
            (setf info 5))
          ((< lda (f2cl-lib:int-add k 1))

```

```

      (setf info 7))
      ((= incx 0)
       (setf info 9)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZTBSV" info)
   (go end_label)))
(if (= n 0) (go end_label))
(setf noconj (char-equal trans #\T))
(setf nounit (char-equal diag #\N))
(cond
  ((<= incx 0)
   (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx))))

  ((/= incx 1)
   (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
    ((char-equal uplo #\U)
     (setf kplus1 (f2cl-lib:int-add k 1))
     (cond
      ((= incx 1)
       (f2cl-lib:fdof (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        ((> j 1) nil)
        (tagbody
         (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
           (setf l (f2cl-lib:int-sub kplus1 j))
           (if nounit
            (setf (f2cl-lib:fref x-%data%
                                (j)
                                ((1 *))
                                x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                                (j)
                                ((1 *))
                                x-%offset%)
                (f2cl-lib:fref a-%data%
                                (kplus1 j)
                                ((1 lda) (1 *))
                                a-%offset%))))
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (f2cl-lib:fdof (i

```



```

(f2cl-lib:int-add j
  (f2cl-lib:int-sub 1))
(f2cl-lib:int-add i
  (f2cl-lib:int-sub 1)))
(> i
  (max (the fixnum 1)
    (the fixnum
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub
          k))))))
nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
        (i)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i)
            j)
            ((1 lda) (1 *))
            a-%offset%))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (setf kx (f2cl-lib:int-sub kx incx))
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (setf ix kx)
        (setf l (f2cl-lib:int-sub kplus1 j))
        (if nounit
          (setf (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                  (jx)

```

```

((1 *))
x-%offset%)
(f2cl-lib:fref a-%data%
  (kplus1 j)
  ((1 lda) (1 *))
  a-%offset%))))
(setf temp
  (f2cl-lib:fref x-%data%
    (jx)
    ((1 *))
    x-%offset%))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  (> i
    (max (the fixnum 1)
      (the fixnum
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub
            k))))))
  nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
    (ix)
    ((1 *))
    x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i)
            j)
          ((1 lda) (1 *))
          a-%offset%))))
    (setf ix (f2cl-lib:int-sub ix incx))))))
(setf jx (f2cl-lib:int-sub jx incx))))))
(t
  (cond
    (= incx 1)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        (/= (f2cl-lib:fref x (j) ((1 *))) zero)
        (setf 1 (f2cl-lib:int-sub 1 j))

```

```

(if nunit
  (setf (f2cl-lib:fref x-%data%
                      (j)
                      ((1 *))
                      x-%offset%)
        (/
          (f2cl-lib:fref x-%data%
                        (j)
                        ((1 *))
                        x-%offset%)
          (f2cl-lib:fref a-%data%
                        (1 j)
                        ((1 lda) (1 *))
                        a-%offset%))))
(setf temp
  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
              (f2cl-lib:int-add i 1))
  (> i
    (min (the fixnum n)
          (the fixnum
            (f2cl-lib:int-add j k))))
  nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%)
        (-
          (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%)
          (* temp
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i)
                           j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf kx (f2cl-lib:int-add kx incx))
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
        (setf ix kx)
        (setf l (f2cl-lib:int-sub 1 j))
        (if nunit

```

```

        (setf (f2cl-lib:fref x-%data%
                           (jx)
                           ((1 *))
                           x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                              (jx)
                              ((1 *))
                              x-%offset%)
                (f2cl-lib:fref a-%data%
                              (1 j)
                              ((1 lda) (1 *))
                              a-%offset%))))
    (setf temp
      (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                  (f2cl-lib:int-add i 1))
      ((> i
        (min (the fixnum n)
              (the fixnum
                (f2cl-lib:int-add j k))))
       nil)
    (tagbody
      (setf (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%)
            (-
              (f2cl-lib:fref x-%data%
                            (ix)
                            ((1 *))
                            x-%offset%)
              (* temp
                (f2cl-lib:fref a-%data%
                              ((f2cl-lib:int-add 1 i)
                               j)
                              ((1 lda) (1 *))
                              a-%offset%))))
      (setf ix (f2cl-lib:int-add ix incx))))))
    (setf jx (f2cl-lib:int-add jx incx))))))
  (t
   (cond
    ((char-equal uplo #\U)
     (setf kplus1 (f2cl-lib:int-add k 1))
     (cond
      ((= incx 1)
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

(> j n) nil)
(tagbody
  (setf temp
    (f2c1-lib:fref x-%data% (j) ((1 *) x-%offset%))
  (setf l (f2c1-lib:int-sub kplus1 j))
  (cond
    (noconj
      (f2c1-lib:fdo (i
        (max (the fixnum 1)
              (the fixnum
                (f2c1-lib:int-add j
                  (f2c1-lib:int-sub
                    k))))
        (f2c1-lib:int-add i 1))
      (> i
        (f2c1-lib:int-add j
          (f2c1-lib:int-sub
            1)))
      nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2c1-lib:fref a-%data%
              ((f2c1-lib:int-add l i)
               j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2c1-lib:fref x-%data%
              (i)
              ((1 *)
               x-%offset%))))))
      (if nounit
        (setf temp
          (/ temp
            (f2c1-lib:fref a-%data%
              (kplus1 j)
              ((1 lda) (1 *))
              a-%offset%))))))
  (t
    (f2c1-lib:fdo (i
      (max (the fixnum 1)
            (the fixnum
              (f2c1-lib:int-add j
                (f2c1-lib:int-sub
                  k))))
      (f2c1-lib:int-add i 1))
    (> i
      (f2c1-lib:int-add j
        (f2c1-lib:int-sub
          k))))

```

```

1)))
      nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                ((f2cl-lib:int-add 1 i)
                  j)
                ((1 lda) (1 *))
                a-%offset%)))
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
      (if nounit
        (setf temp
          (/ temp
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (kplus1 j)
                ((1 lda) (1 *))
                a-%offset%))))))
        (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
              temp))))
  (t
    (setf jx kx)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix kx)
      (setf l (f2cl-lib:int-sub kplus1 j))
      (cond
        (noconj
          (f2cl-lib:fdo (i
            (max (the fixnum 1)
              (the fixnum
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub
                    k))))
            (f2cl-lib:int-add i 1))
          ((> i
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub
                1)))
            nil)
          (tagbody

```

```

      (setf temp
        (- temp
          (*
            (f2cl-lib:fref a-%data%
                          ((f2cl-lib:int-add 1 i)
                           j)
                          ((1 lda) (1 *)))
            a-%offset%)
            (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%))))
      (setf ix (f2cl-lib:int-add ix incx)))
    (if nunit
      (setf temp
        (/ temp
          (f2cl-lib:fref a-%data%
                        (kplus1 j)
                        ((1 lda) (1 *)))
          a-%offset%))))
  (t
   (f2cl-lib:fdo (i
                  (max (the fixnum 1)
                       (the fixnum
                        (f2cl-lib:int-add j
                                           (f2cl-lib:int-sub
                                            k))))
                  (f2cl-lib:int-add i 1))
     (> i
      (f2cl-lib:int-add j
                        (f2cl-lib:int-sub
                         1)))
      nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                            ((f2cl-lib:int-add 1 i)
                             j)
                            ((1 lda) (1 *)))
              a-%offset%)
            (f2cl-lib:fref x-%data%
                          (ix)
                          ((1 *))
                          x-%offset%))))
      (setf ix (f2cl-lib:int-add ix incx)))
    (if nunit
      (setf temp

```

```

                                (/ temp
                                (f2cl-lib:dconjg
                                 (f2cl-lib:fref a-%data%
                                                  (kplus1 j)
                                                  ((1 lda) (1 *))
                                                  a-%offset%))))))
                                (setf (f2cl-lib:fref x-%data% (jx) ((1 *) x-%offset%)
                                              temp)
                                (setf jx (f2cl-lib:int-add jx incx))
                                (if (> j k) (setf kx (f2cl-lib:int-add kx incx))))))
(t
 (cond
  ((= incx 1)
   (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                  (> j 1) nil)
   (tagbody
    (setf temp
            (f2cl-lib:fref x-%data% (j) ((1 *) x-%offset%))
            (setf l (f2cl-lib:int-sub 1 j))
            (cond
             (noconj
              (f2cl-lib:fdo (i
                             (min (the fixnum n)
                                   (the fixnum
                                     (f2cl-lib:int-add j k)))
                             (f2cl-lib:int-add i
                                                  (f2cl-lib:int-sub 1)))
                            (> i (f2cl-lib:int-add j 1)) nil)
              (tagbody
               (setf temp
                       (- temp
                          (*
                           (f2cl-lib:fref a-%data%
                                           ((f2cl-lib:int-add l i)
                                           j)
                                           ((1 lda) (1 *))
                                           a-%offset%)
                           (f2cl-lib:fref x-%data%
                                           (i)
                                           ((1 *))
                                           x-%offset%))))))
              (if nounit
               (setf temp
                       (/ temp
                          (f2cl-lib:fref a-%data%
                                           (1 j)
                                           ((1 lda) (1 *))
                                           a-%offset%))))))
              (t
               (f2cl-lib:fdo (i

```



```

(min (the fixnum n)
    (the fixnum
      (f2cl-lib:int-add j k)))
(f2cl-lib:int-add i
  (f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1)) nil)
(tagbody
  (setf temp
    (- temp
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
                        ((f2cl-lib:int-add 1 i)
                          j)
                        ((1 lda) (1 *))
                        a-%offset%))
        (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%))))))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:dconjg
        (f2cl-lib:fref a-%data%
                        (1 j)
                        ((1 lda) (1 *))
                        a-%offset%))))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp)))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix kx)
    (setf l (f2cl-lib:int-sub 1 j))
    (cond
      (noconj
        (f2cl-lib:fdo (i
          (min (the fixnum n)
              (the fixnum
                (f2cl-lib:int-add j k)))

```

```

(f2cl-lib:int-add i
  (f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1)) nil)
(tagbody
  (setf temp
    (- temp
      (*
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add 1 i)
           j)
          ((1 lda) (1 *))
          a-%offset%))
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))))
  (setf ix (f2cl-lib:int-sub ix incx))))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (1 j)
        ((1 lda) (1 *))
        a-%offset%))))))
(t
  (f2cl-lib:fdo (i
    (min (the fixnum n)
          (the fixnum
            (f2cl-lib:int-add j k)))
    (f2cl-lib:int-add i
      (f2cl-lib:int-sub 1)))
    (> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              ((f2cl-lib:int-add 1 i)
               j)
              ((1 lda) (1 *))
              a-%offset%))
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))))
      (setf ix (f2cl-lib:int-sub ix incx))))
  (if nunit
    (setf temp
      (/ temp

```

```

                                (f2cl-lib:dconjg
                                (f2cl-lib:fref a-%data%
                                              (1 j)
                                              ((1 lda) (1 *))
                                              a-%offset%))))))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
            temp)
      (setf jx (f2cl-lib:int-sub jx incx))
      (if (>= (f2cl-lib:int-sub n j) k)
          (setf kx (f2cl-lib:int-sub kx incx)))))))))
end_label
  (return (values nil nil nil nil nil nil nil nil))))

```

ztpmv BLAS

— ztpmv.input —

```

)set break resume
)sys rm -f ztpmv.output
)spool ztpmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztpmv.help —

```

=====
ztpmv examples
=====

=====
Man Page Details
=====

```

NAME

ZTPMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$, or $x := \text{conjg}(A')*x$,

SYNOPSIS

```
SUBROUTINE ZTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )
```

```
    INTEGER      INCX, N
```

```
    CHARACTER*1  DIAG, TRANS, UPLO
```

```
    COMPLEX*16   AP( * ), X( * )
```

PURPOSE

ZTPMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := Ax$.

TRANS = 'T' or 't' $x := A^T x$.

TRANS = 'C' or 'c' $x := \text{conjg}(A) x$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

- N - INTEGER.
On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.
- AP - COMPLEX*16 array of DIMENSION at least
((n*(n + 1))/2). Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity. Unchanged on exit.
- X - COMPLEX*16 array of dimension at least
(1 + (n - 1)*abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.
- INCX - INTEGER.
On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— ztpmv.f —

```

SUBROUTINE ZTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )
*   .. Scalar Arguments ..
    INTEGER          INCX, N
    CHARACTER*1      DIAG, TRANS, UPLO
*   .. Array Arguments ..
    COMPLEX*16       AP( * ), X( * )
*   ..
*
*   Level 2 Blas routine.
*
*   -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.

```

```

*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.
*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
      COMPLEX*16          ZERO
      PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      .. Local Scalars ..
      COMPLEX*16          TEMP
      INTEGER             I, INFO, IX, J, JX, K, KK, KX
      LOGICAL             NOCONJ, NOUNIT
*
*      .. External Functions ..
      LOGICAL             LSAME
      EXTERNAL            LSAME
*
*      .. External Subroutines ..
      EXTERNAL            XERBLA
*
*      .. Intrinsic Functions ..
      INTRINSIC            DCONJG
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$           .NOT.LSAME( UPLO , 'L' ) ) THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$           .NOT.LSAME( TRANS, 'T' ).AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$           .NOT.LSAME( DIAG , 'N' ) ) THEN
          INFO = 3
      ELSE IF( N.LT.0 ) THEN
          INFO = 4
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 7
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZTPMV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*

```

```

      NOCONJ = LSAME( TRANS, 'T' )
      NOUNIT = LSAME( DIAG, 'N' )

*
*   Set up the start point in X if the increment is not unity. This
*   will be ( N - 1 ) * INCX too small for descending loops.
*
      IF( INCX.LE.0 ) THEN
        KX = 1 - ( N - 1 ) * INCX
      ELSE IF( INCX.NE.1 ) THEN
        KX = 1
      END IF

*
*   Start the operations. In this version the elements of AP are
*   accessed sequentially with one pass through AP.
*
      IF( LSAME( TRANS, 'N' ) ) THEN

*
*       Form  x := A*x.
*
        IF( LSAME( UPLO, 'U' ) ) THEN
          KK = 1
          IF( INCX.EQ.1 ) THEN
            DO 20, J = 1, N
              IF( X( J ).NE.ZERO ) THEN
                TEMP = X( J )
                K = KK
                DO 10, I = 1, J - 1
                  X( I ) = X( I ) + TEMP*AP( K )
                  K = K + 1
                10          CONTINUE
              IF( NOUNIT )
                $          X( J ) = X( J ) * AP( KK + J - 1 )
              END IF
              KK = KK + J
            20          CONTINUE
          ELSE
            JX = KX
            DO 40, J = 1, N
              IF( X( JX ).NE.ZERO ) THEN
                TEMP = X( JX )
                IX = KX
                DO 30, K = KK, KK + J - 2
                  X( IX ) = X( IX ) + TEMP*AP( K )
                  IX = IX + INCX
                30          CONTINUE
              IF( NOUNIT )
                $          X( JX ) = X( JX ) * AP( KK + J - 1 )
              END IF
              JX = JX + INCX
              KK = KK + J
            40          CONTINUE
          END IF
        ELSE
          IF( LSAME( UPLO, 'L' ) ) THEN
            KK = N
            IF( INCX.EQ.1 ) THEN
              DO 20, J = N, 1, -1
                IF( X( J ).NE.ZERO ) THEN
                  TEMP = X( J )
                  K = KK
                  DO 10, I = J + 1, N
                    X( I ) = X( I ) + TEMP*AP( K )
                    K = K + 1
                  10          CONTINUE
                IF( NOUNIT )
                  $          X( J ) = X( J ) * AP( KK + J - 1 )
                END IF
                KK = KK + J
              20          CONTINUE
            ELSE
              JX = KX
              DO 40, J = N, 1, -1
                IF( X( JX ).NE.ZERO ) THEN
                  TEMP = X( JX )
                  IX = KX
                  DO 30, K = KK, KK + J - 2
                    X( IX ) = X( IX ) + TEMP*AP( K )
                    IX = IX + INCX
                  30          CONTINUE
                IF( NOUNIT )
                  $          X( JX ) = X( JX ) * AP( KK + J - 1 )
                END IF
                JX = JX + INCX
                KK = KK + J
              40          CONTINUE
            END IF
          ELSE
            IF( INCX.EQ.1 ) THEN
              DO 20, J = 1, N
                IF( X( J ).NE.ZERO ) THEN
                  TEMP = X( J )
                  K = KK
                  DO 10, I = 1, J - 1
                    X( I ) = X( I ) + TEMP*AP( K )
                    K = K + 1
                  10          CONTINUE
                IF( NOUNIT )
                  $          X( J ) = X( J ) * AP( KK + J - 1 )
                END IF
                KK = KK + J
              20          CONTINUE
            ELSE
              JX = KX
              DO 40, J = 1, N
                IF( X( JX ).NE.ZERO ) THEN
                  TEMP = X( JX )
                  IX = KX
                  DO 30, K = KK, KK + J - 2
                    X( IX ) = X( IX ) + TEMP*AP( K )
                    IX = IX + INCX
                  30          CONTINUE
                IF( NOUNIT )
                  $          X( JX ) = X( JX ) * AP( KK + J - 1 )
                END IF
                JX = JX + INCX
                KK = KK + J
              40          CONTINUE
            END IF
          END IF
        END IF
      END IF

```

```

40      CONTINUE
      END IF
    ELSE
      KK = ( N*( N + 1 ) )/2
      IF( INCX.EQ.1 )THEN
        DO 60, J = N, 1, -1
          IF( X( J ).NE.ZERO )THEN
            TEMP = X( J )
            K = KK
            DO 50, I = N, J + 1, -1
              X( I ) = X( I ) + TEMP*AP( K )
              K = K - 1
50          CONTINUE
            IF( NUNIT )
              $      X( J ) = X( J )*AP( KK - N + J )
            END IF
            KK = KK - ( N - J + 1 )
60      CONTINUE
    ELSE
      KX = KX + ( N - 1 )*INCX
      JX = KX
      DO 80, J = N, 1, -1
        IF( X( JX ).NE.ZERO )THEN
          TEMP = X( JX )
          IX = KX
          DO 70, K = KK, KK - ( N - ( J + 1 ) ), -1
            X( IX ) = X( IX ) + TEMP*AP( K )
            IX = IX - INCX
70          CONTINUE
            IF( NUNIT )
              $      X( JX ) = X( JX )*AP( KK - N + J )
            END IF
            JX = JX - INCX
            KK = KK - ( N - J + 1 )
80      CONTINUE
    END IF
  END IF
ELSE
*
*      Form x := A'*x or x := conjg( A' )*x.
*
      IF( LSAME( UPLO, 'U' ) )THEN
        KK = ( N*( N + 1 ) )/2
        IF( INCX.EQ.1 )THEN
          DO 110, J = N, 1, -1
            TEMP = X( J )
            K = KK - 1
            IF( NOCONJ )THEN
              IF( NUNIT )
                $      TEMP = TEMP*AP( KK )
            
```



```

DO 90, I = J - 1, 1, -1
    TEMP = TEMP + AP( K ) * X( I )
    K = K - 1
90    CONTINUE
ELSE
    IF( NUNIT )
        $    TEMP = TEMP * DCONJG( AP( KK ) )
        DO 100, I = J - 1, 1, -1
            TEMP = TEMP + DCONJG( AP( K ) ) * X( I )
            K = K - 1
100    CONTINUE
        END IF
        X( J ) = TEMP
        KK = KK - J
110    CONTINUE
ELSE
    JX = KX + ( N - 1 ) * INCX
    DO 140, J = N, 1, -1
        TEMP = X( JX )
        IX = JX
        IF( NOCONJ ) THEN
            IF( NUNIT )
                $    TEMP = TEMP * AP( KK )
            DO 120, K = KK - 1, KK - J + 1, -1
                IX = IX - INCX
                TEMP = TEMP + AP( K ) * X( IX )
120    CONTINUE
            ELSE
                IF( NUNIT )
                    $    TEMP = TEMP * DCONJG( AP( KK ) )
                DO 130, K = KK - 1, KK - J + 1, -1
                    IX = IX - INCX
                    TEMP = TEMP + DCONJG( AP( K ) ) * X( IX )
130    CONTINUE
                END IF
                X( JX ) = TEMP
                JX = JX - INCX
                KK = KK - J
140    CONTINUE
            END IF
        ELSE
            KK = 1
            IF( INCX.EQ.1 ) THEN
                DO 170, J = 1, N
                    TEMP = X( J )
                    K = KK + 1
                    IF( NOCONJ ) THEN
                        IF( NUNIT )
                            $    TEMP = TEMP * AP( KK )
                        DO 150, I = J + 1, N

```

```

        TEMP = TEMP + AP( K ) * X( I )
        K = K + 1
150      CONTINUE
      ELSE
        IF( NUNIT )
          $      TEMP = TEMP * DCONJG( AP( KK ) )
          DO 160, I = J + 1, N
            TEMP = TEMP + DCONJG( AP( K ) ) * X( I )
            K = K + 1
160      CONTINUE
        END IF
        X( J ) = TEMP
        KK = KK + ( N - J + 1 )
170      CONTINUE
      ELSE
        JX = KX
        DO 200, J = 1, N
          TEMP = X( JX )
          IX = JX
          IF( NOCONJ ) THEN
            IF( NUNIT )
              $      TEMP = TEMP * AP( KK )
              DO 180, K = KK + 1, KK + N - J
                IX = IX + INCX
                TEMP = TEMP + AP( K ) * X( IX )
180            CONTINUE
              ELSE
                IF( NUNIT )
                  $      TEMP = TEMP * DCONJG( AP( KK ) )
                  DO 190, K = KK + 1, KK + N - J
                    IX = IX + INCX
                    TEMP = TEMP + DCONJG( AP( K ) ) * X( IX )
190                CONTINUE
              END IF
              X( JX ) = TEMP
              JX = JX + INCX
              KK = KK + ( N - J + 1 )
200            CONTINUE
          END IF
        END IF
      END IF
*
      RETURN
*
*      End of ZTPMV .
*
      END

```

```

(setf kx 1)))
(cond
  ((char-equal trans #\N)
    (cond
      ((char-equal uplo #\U)
        (setf kk 1)
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                  (setf temp
                    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
                  (setf k kk)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i
                      (f2cl-lib:int-add j
                        (f2cl-lib:int-sub
                          1))))
                    nil)
                  (tagbody
                    (setf (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%)
                      (+
                        (f2cl-lib:fref x-%data%
                          (i)
                          ((1 *))
                          x-%offset%)
                        (* temp
                          (f2cl-lib:fref ap-%data%
                            (k)
                            ((1 *))
                            ap-%offset%))))
                    (setf k (f2cl-lib:int-add k 1))))
                  (if nounit
                    (setf (f2cl-lib:fref x-%data%
                      (j)
                      ((1 *))
                      x-%offset%)
                      (*
                        (f2cl-lib:fref x-%data%
                          (j)
                          ((1 *))
                          x-%offset%)
                        (f2cl-lib:fref ap-%data%
                          ((f2cl-lib:int-sub

```

```

                                (f2cl-lib:int-add kk j)
                                1))
                                ((1 *))
                                ap-%offset%))))))
    (setf kk (f2cl-lib:int-add kk j))))
  (t
   (setf jx kx)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  (> j n) nil)
   (tagbody
    (cond
     ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
      (setf temp
              (f2cl-lib:fref x-%data%
                              (jx)
                              ((1 *))
                              x-%offset%))

      (setf ix kx)
      (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
                    (> k
                     (f2cl-lib:int-add kk
                                          j
                                          (f2cl-lib:int-sub
                                           2)))
                     nil)
      (tagbody
       (setf (f2cl-lib:fref x-%data%
                             (ix)
                             ((1 *))
                             x-%offset%)
              (+
               (f2cl-lib:fref x-%data%
                               (ix)
                               ((1 *))
                               x-%offset%)
               (* temp
                  (f2cl-lib:fref ap-%data%
                                   (k)
                                   ((1 *))
                                   ap-%offset%))))
              (setf ix (f2cl-lib:int-add ix incx))))
      (if nunit
         (setf (f2cl-lib:fref x-%data%
                               (jx)
                               ((1 *))
                               x-%offset%)
                (*
                 (f2cl-lib:fref x-%data%
                                 (jx)
                                 ((1 *))

```

```

                                x-%offset%)
(f2cl-lib:fref ap-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add kk j)
    1))
  ((1 *))
  ap-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
            (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
            (setf k kk)
            (f2cl-lib:fdo (i n
              (f2cl-lib:int-add i
                (f2cl-lib:int-sub 1)))
              (> i (f2cl-lib:int-add j 1)) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
                x-%offset%)
                (+
                  (f2cl-lib:fref x-%data%
                    (i)
                    ((1 *))
                    x-%offset%)
                  (* temp
                    (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%))))
                (setf k (f2cl-lib:int-sub k 1))))
            (if nounit
              (setf (f2cl-lib:fref x-%data%
                (j)
                ((1 *))
                x-%offset%)
                (*
                  (f2cl-lib:fref x-%data%
                    (j)
                    ((1 *))

```

```

                                x-%offset%)
(f2cl-lib:fref ap-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub kk n)
    j))
  ((1 *))
  ap-%offset%))))))
(setf kk
  (f2cl-lib:int-sub kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (setf jx kx)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j 1) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (setf temp
            (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%))
          (setf ix kx)
          (f2cl-lib:fdo (k kk
            (f2cl-lib:int-add k
              (f2cl-lib:int-sub 1)))
            ((> k
              (f2cl-lib:int-add kk
                (f2cl-lib:int-sub
                  (f2cl-lib:int-add
                    n
                    (f2cl-lib:int-sub
                      (f2cl-lib:int-add
                        j
                        1))))))
              nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data%
                (ix)
                ((1 *))
                x-%offset%)
                (+
                  (f2cl-lib:fref x-%data%

```

```

                                (ix)
                                ((1 *))
                                x-%offset%)
      (* temp
        (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)))
      (setf ix (f2cl-lib:int-sub ix incx)))
    (if nounit
      (setf (f2cl-lib:fref x-%data%
                          (jx)
                          ((1 *))
                          x-%offset%)
            (*
              (f2cl-lib:fref x-%data%
                            (jx)
                            ((1 *))
                            x-%offset%)
              (f2cl-lib:fref ap-%data%
                            ((f2cl-lib:int-add
                              (f2cl-lib:int-sub kk n)
                              j))
                            ((1 *))
                            ap-%offset%))))))
      (setf jx (f2cl-lib:int-sub jx incx))
      (setf kk
        (f2cl-lib:int-sub kk
          (f2cl-lib:int-add
            (f2cl-lib:int-sub n j)
            1))))))
  (t
    (cond
      ((char-equal uplo #\U)
        (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                          (> j 1) nil)
            (tagbody
              (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
              (setf k (f2cl-lib:int-sub kk 1))
              (cond
                (noconj
                  (if nounit
                    (setf temp
                      (* temp
                        (f2cl-lib:fref ap-%data%
                                      (kk)

```



```

((1 *))
ap-%offset%)))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  (> i 1) nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%))))
    (setf k (f2cl-lib:int-sub k 1)))))
(t
  (if nount
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref ap-%data%
            (kk)
            ((1 *))
            ap-%offset%))))
    (f2cl-lib:fdo (i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub 1))
      (f2cl-lib:int-add i
        (f2cl-lib:int-sub 1)))
      (> i 1) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%)
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))
          (setf k (f2cl-lib:int-sub k 1)))))

```

```

      (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
            temp)
      (setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix jx)
      (cond
        (noconj
          (if nounit
            (setf temp
              (* temp
                (f2cl-lib:fref ap-%data%
                  (kk)
                  ((1 *))
                  ap-%offset%))))
            (f2cl-lib:fdo (k
              (f2cl-lib:int-add kk
                (f2cl-lib:int-sub 1))
              (f2cl-lib:int-add k
                (f2cl-lib:int-sub 1)))
              (> k
                (f2cl-lib:int-add kk
                  (f2cl-lib:int-sub
                    j
                    1))
                nil)
              (tagbody
                (setf ix (f2cl-lib:int-sub ix incx))
                (setf temp
                  (+ temp
                    (*
                      (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%)
                      (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%))))))))
            (t
              (if nounit

```

```

      (setf temp
        (* temp
          (f2cl-lib:dconjg
            (f2cl-lib:fref ap-%data%
                          (kk)
                          ((1 *))
                          ap-%offset%))))))
(f2cl-lib:fdo (k
              (f2cl-lib:int-add kk
                                (f2cl-lib:int-sub 1))
              (f2cl-lib:int-add k
                                (f2cl-lib:int-sub 1)))
  (> k
    (f2cl-lib:int-add kk
                      (f2cl-lib:int-sub
                        j)
                      1))
    nil)
(tagbody
  (setf ix (f2cl-lib:int-sub ix incx))
  (setf temp
    (+ temp
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref ap-%data%
                        (k)
                        ((1 *))
                        ap-%offset%))
        (f2cl-lib:fref x-%data%
                      (ix)
                      ((1 *))
                      x-%offset%))))))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-sub jx incx))
  (setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf kk 1)
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf k (f2cl-lib:int-add kk 1))
        (cond
          (noconj
            (if nunit
              (setf temp

```

```

(* temp
  (f2cl-lib:fref ap-%data%
                (kk)
                ((1 *))
                ap-%offset%))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                (f2cl-lib:int-add i 1))
              (> i n) nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref ap-%data%
                      (k)
                      ((1 *))
                      ap-%offset%)
        (f2cl-lib:fref x-%data%
                      (i)
                      ((1 *))
                      x-%offset%))))
    (setf k (f2cl-lib:int-add k 1))))))
(t
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref ap-%data%
                        (kk)
                        ((1 *))
                        ap-%offset%))))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                    (f2cl-lib:int-add i 1))
                  (> i n) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref ap-%data%
                            (k)
                            ((1 *))
                            ap-%offset%))
            (f2cl-lib:fref x-%data%
                            (i)
                            ((1 *))
                            x-%offset%))))
          (setf k (f2cl-lib:int-add k 1))))))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
          temp)
    (setf kk

```



```

                                ap-%offset%))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add kk 1)
               (f2cl-lib:int-add k 1))
              (> k
               (f2cl-lib:int-add kk
                                   n
               (f2cl-lib:int-sub
                j))))
              nil)
(tagbody
 (setf ix (f2cl-lib:int-add ix incx))
 (setf temp
  (+ temp
     (*
      (f2cl-lib:dconjg
       (f2cl-lib:fref ap-%data%
                       (k)
                       ((1 *))
                       ap-%offset%))
      (f2cl-lib:fref x-%data%
                     (ix)
                     ((1 *))
                     x-%offset%))))))
 (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
       temp)
 (setf jx (f2cl-lib:int-add jx incx))
 (setf kk
  (f2cl-lib:int-add kk
                    (f2cl-lib:int-add
                     (f2cl-lib:int-sub n j)
                     1))))))
end_label
(return (values nil nil nil nil nil nil nil))))

```

ztpsv BLAS

— ztpsv.input —

```

)set break resume
)sys rm -f ztpsv.output
)spool ztpsv.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— ztpsv.help —

```
=====
ztpsv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZTPSV - solve one of the systems of equations $Ax = b$, or $A'x = b$, or $\text{conjg}(A')x = b$,

SYNOPSIS

SUBROUTINE ZTPSV (UPLO, TRANS, DIAG, N, AP, X, INCX)

INTEGER INCX, N

CHARACTER*1 DIAG, TRANS, UPLO

COMPLEX*16 AP(*), X(*)

PURPOSE

ZTPSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $A*x = b$.

TRANS = 'T' or 't' $A'*x = b$.

TRANS = 'C' or 'c' $\text{conjg}(A)*x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

AP - COMPLEX*16 array of DIMENSION at least $((n*(n+1))/2)$. Before entry with UPLO = 'U' or 'u', the array AP must contain the upper triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(1, 2) and a(2, 2) respectively, and so on. Before entry with UPLO = 'L' or 'l', the array AP must contain the lower triangular matrix packed sequentially, column by column, so that AP(1) contains a(1, 1), AP(2) and AP(3) contain a(2, 1) and a(3, 1) respectively, and so on. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n-1)*\text{abs}(\text{INCX}))$. Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.

INCX - INTEGER.
 On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— ztpsv.f —

```

SUBROUTINE ZTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )
*
* .. Scalar Arguments ..
  INTEGER          INCX, N
  CHARACTER*1      DIAG, TRANS, UPLO
*
* .. Array Arguments ..
  COMPLEX*16       AP( * ), X( * )
*
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*
* .. Parameters ..
  COMPLEX*16       ZERO
  PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* .. Local Scalars ..
  COMPLEX*16       TEMP
  INTEGER          I, INFO, IX, J, JX, K, KK, KX
  LOGICAL          NOCONJ, NOUNIT
*
* .. External Functions ..
  LOGICAL          LSAME
  EXTERNAL         LSAME
*
* .. External Subroutines ..
  EXTERNAL         XERBLA
*
* .. Intrinsic Functions ..
  INTRINSIC        DCONJG
*
* ..
*
* .. Executable Statements ..
*
* Test the input parameters.
*
  INFO = 0
  IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
```

```

$      .NOT.LSAME( UPLO , 'L' )      )THEN
      INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$      .NOT.LSAME( TRANS, 'T' ).AND.
$      .NOT.LSAME( TRANS, 'C' )      )THEN
      INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$      .NOT.LSAME( DIAG , 'N' )      )THEN
      INFO = 3
      ELSE IF( N.LT.0 )THEN
      INFO = 4
      ELSE IF( INCX.EQ.0 )THEN
      INFO = 7
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZTPSV ', INFO )
        RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
      NOCONJ = LSAME( TRANS, 'T' )
      NOUNIT = LSAME( DIAG , 'N' )
*
*      Set up the start point in X if the increment is not unity. This
*      will be ( N - 1 )*INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 )*INCX
      ELSE IF( INCX.NE.1 )THEN
        KX = 1
      END IF
*
*      Start the operations. In this version the elements of AP are
*      accessed sequentially with one pass through AP.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form  $x := inv(A)x$ .
*
      IF( LSAME( UPLO, 'U' ) )THEN
        KK = ( N*( N + 1 ) )/2
        IF( INCX.EQ.1 )THEN
          DO 20, J = N, 1, -1
            IF( X( J ).NE.ZERO )THEN
              IF( NOUNIT )
$              X( J ) = X( J )/AP( KK )

```

```

        TEMP = X( J )
        K     = KK     - 1
        DO 10, I = J - 1, 1, -1
            X( I ) = X( I ) - TEMP*AP( K )
            K     = K     - 1
10      CONTINUE
        END IF
        KK = KK - J
20      CONTINUE
    ELSE
        JX = KX + ( N - 1 )*INCX
        DO 40, J = N, 1, -1
            IF( X( JX ).NE.ZERO ) THEN
                IF( NOUNIT )
$           X( JX ) = X( JX )/AP( KK )
                TEMP = X( JX )
                IX   = JX
                DO 30, K = KK - 1, KK - J + 1, -1
                    IX   = IX   - INCX
                    X( IX ) = X( IX ) - TEMP*AP( K )
30              CONTINUE
                END IF
                JX = JX - INCX
                KK = KK - J
40              CONTINUE
            END IF
        ELSE
            KK = 1
            IF( INCX.EQ.1 ) THEN
                DO 60, J = 1, N
                    IF( X( J ).NE.ZERO ) THEN
                        IF( NOUNIT )
$                   X( J ) = X( J )/AP( KK )
                        TEMP = X( J )
                        K     = KK     + 1
                        DO 50, I = J + 1, N
                            X( I ) = X( I ) - TEMP*AP( K )
                            K     = K     + 1
50                      CONTINUE
                        END IF
                        KK = KK + ( N - J + 1 )
60                      CONTINUE
                    ELSE
                        JX = KX
                        DO 80, J = 1, N
                            IF( X( JX ).NE.ZERO ) THEN
                                IF( NOUNIT )
$                               X( JX ) = X( JX )/AP( KK )
                                TEMP = X( JX )
                                IX   = JX

```

```

DO 70, K = KK + 1, KK + N - J
    IX      = IX      + INCX
    X( IX ) = X( IX ) - TEMP*AP( K )
70    CONTINUE
    END IF
    JX = JX + INCX
    KK = KK + ( N - J + 1 )
80    CONTINUE
    END IF
    END IF
ELSE
*
*    Form x := inv( A' ) * x or x := inv( conjg( A' ) ) * x.
*
    IF( LSAME( UPLO, 'U' ) ) THEN
        KK = 1
        IF( INCX.EQ.1 ) THEN
            DO 110, J = 1, N
                TEMP = X( J )
                K = KK
                IF( NOCONJ ) THEN
                    DO 90, I = 1, J - 1
                        TEMP = TEMP - AP( K ) * X( I )
                        K = K + 1
90                    CONTINUE
                    IF( NOUNIT )
$                        TEMP = TEMP/AP( KK + J - 1 )
                    ELSE
                        DO 100, I = 1, J - 1
                            TEMP = TEMP - DCONJG( AP( K ) ) * X( I )
                            K = K + 1
100                   CONTINUE
                    IF( NOUNIT )
$                        TEMP = TEMP/DCONJG( AP( KK + J - 1 ) )
                    END IF
                    X( J ) = TEMP
                    KK = KK + J
110                CONTINUE
            ELSE
                JX = KX
                DO 140, J = 1, N
                    TEMP = X( JX )
                    IX = KX
                    IF( NOCONJ ) THEN
                        DO 120, K = KK, KK + J - 2
                            TEMP = TEMP - AP( K ) * X( IX )
                            IX = IX + INCX
120                        CONTINUE
                    IF( NOUNIT )
$                        TEMP = TEMP/AP( KK + J - 1 )

```

```

        ELSE
            DO 130, K = KK, KK + J - 2
                TEMP = TEMP - DCONJG( AP( K ) ) * X( IX )
                IX = IX + INCX
130          CONTINUE
            IF( NUNIT )
                $      TEMP = TEMP / DCONJG( AP( KK + J - 1 ) )
            END IF
            X( JX ) = TEMP
            JX = JX + INCX
            KK = KK + J
140          CONTINUE
        END IF
    ELSE
        KK = ( N * ( N + 1 ) ) / 2
        IF( INCX.EQ.1 ) THEN
            DO 170, J = N, 1, -1
                TEMP = X( J )
                K = KK
                IF( NOCONJ ) THEN
                    DO 150, I = N, J + 1, -1
                        TEMP = TEMP - AP( K ) * X( I )
                        K = K - 1
150                  CONTINUE
                    IF( NUNIT )
                        $      TEMP = TEMP / AP( KK - N + J )
                    ELSE
                        DO 160, I = N, J + 1, -1
                            TEMP = TEMP - DCONJG( AP( K ) ) * X( I )
                            K = K - 1
160                  CONTINUE
                        IF( NUNIT )
                            $      TEMP = TEMP / DCONJG( AP( KK - N + J ) )
                        END IF
                        X( J ) = TEMP
                        KK = KK - ( N - J + 1 )
170                  CONTINUE
                ELSE
                    KX = KX + ( N - 1 ) * INCX
                    JX = KX
                    DO 200, J = N, 1, -1
                        TEMP = X( JX )
                        IX = KX
                        IF( NOCONJ ) THEN
                            DO 180, K = KK, KK - ( N - ( J + 1 ) ), -1
                                TEMP = TEMP - AP( K ) * X( IX )
                                IX = IX - INCX
180                            CONTINUE
                        IF( NUNIT )
                            $      TEMP = TEMP / AP( KK - N + J )

```

```

ELSE
  DO 190, K = KK, KK - ( N - ( J + 1 ) ), -1
    TEMP = TEMP - DCONJG( AP( K ) ) * X( IX )
    IX = IX - INCX
190    CONTINUE
    IF( NOUNIT )
      $      TEMP = TEMP / DCONJG( AP( KK - N + J ) )
    END IF
    X( JX ) = TEMP
    JX = JX - INCX
    KK = KK - ( N - J + 1 )
200    CONTINUE
  END IF
END IF
END IF
*
RETURN
*
* End of ZTPSV .
*
END

```

— BLAS 2 ztpsv —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztpsv (uplo trans diag n ap x incx)
    (declare (type (simple-array (complex double-float) (*)) x ap)
      (type fixnum incx n)
      (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (ap (complex double-float) ap-%data% ap-%offset%)
       (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0) (k 0)
        (kk 0) (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
          (type fixnum i info ix j jx k kk kx)
          (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
            (setf info 1))
          ((and (not (char-equal trans #\N))

```

```

        (not (char-equal trans #\T))
        (not (char-equal trans #\C)))
    (setf info 2))
  ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
    (setf info 3))
  ((< n 0)
    (setf info 4))
  ((= incx 0)
    (setf info 7)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "ZTPSV" info)
    (go end_label)))
(if (= n 0) (go end_label))
(setf noconj (char-equal trans #\T))
(setf nunit (char-equal diag #\N))
(cond
  ((<= incx 0)
    (setf kx
      (f2cl-lib:int-sub 1
        (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
          incx))))

  ((/= incx 1)
    (setf kx 1)))
(cond
  ((char-equal trans #\N)
    (cond
      ((char-equal uplo #\U)
        (setf kk (the fixnum (truncate (* n (+ n 1)) 2)))
        (cond
          ((= incx 1)
            (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j 1) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
                  (if nunit
                    (setf (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                      (/
                        (f2cl-lib:fref x-%data%
                                          (j)
                                          ((1 *))
                                          x-%offset%)
                        (f2cl-lib:fref ap-%data%
                                          (kk)

```

```

((1 *))
ap-%offset%)))
(setf temp
  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(setf k (f2cl-lib:int-sub kk 1))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  ((> i 1) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
      (i)
      ((1 *))
      x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)
        (* temp
          (f2cl-lib:fref ap-%data%
            (k)
            ((1 *))
            ap-%offset%))))))
    (setf k (f2cl-lib:int-sub k 1))))))
(setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      ((> j 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (jx) ((1 *)) zero)
            (if nounit
              (setf (f2cl-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%)
                (/
                  (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)
                  (f2cl-lib:fref ap-%data%
                    (k)
                    ((1 *))
                    ap-%offset%))))))
            t
            (setf k (f2cl-lib:int-sub k 1))))))
          (setf kk (f2cl-lib:int-sub kk j))))))
  )
)

```



```

                                (kk)
                                ((1 *))
                                ap-%offset%))))
(setf temp
  (f2cl-lib:fref x-%data%
    (jx)
    ((1 *))
    x-%offset%))
(setf ix jx)
(f2cl-lib:fdo (k
  (f2cl-lib:int-add kk
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add k
    (f2cl-lib:int-sub 1))
  ((> k
    (f2cl-lib:int-add kk
      (f2cl-lib:int-sub
        j)
        1))
    nil)
  (tagbody
    (setf ix (f2cl-lib:int-sub ix incx))
    (setf (f2cl-lib:fref x-%data%
      (ix)
      ((1 *))
      x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%)
        (* temp
          (f2cl-lib:fref ap-%data%
            (k)
            ((1 *))
            ap-%offset%))))))
  (setf jx (f2cl-lib:int-sub jx incx))
  (setf kk (f2cl-lib:int-sub kk j))))))
(t
  (setf kk 1)
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
              (if nunit
                (setf (f2cl-lib:fref x-%data%
                  (j)

```

```

((1 *))
x-%offset%)

(/
(f2cl-lib:fref x-%data%
  (j)
  ((1 *))
  x-%offset%)
(f2cl-lib:fref ap-%data%
  (kk)
  ((1 *))
  ap-%offset%)))))
(setf temp
  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(setf k (f2cl-lib:int-add kk 1))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
    (i)
    ((1 *))
    x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
        (i)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)))))
    (setf k (f2cl-lib:int-add k 1))))))
(setf kk
  (f2cl-lib:int-add kk
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n j)
      1))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
        (if nount
          (setf (f2cl-lib:fref x-%data%
            (jx)
            ((1 *))
            x-%offset%)

```

```

(/
  (f2cl-lib:fref x-%data%
                 (jx)
                 ((1 *))
                 x-%offset%)
  (f2cl-lib:fref ap-%data%
                 (kk)
                 ((1 *))
                 ap-%offset%))))
(setf temp
  (f2cl-lib:fref x-%data%
                 (jx)
                 ((1 *))
                 x-%offset%))
(setf ix jx)
(f2cl-lib:fdof (k (f2cl-lib:int-add kk 1)
                  (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add kk
                        n
                        (f2cl-lib:int-sub
                          j))))
  nil)
(tagbody
  (setf ix (f2cl-lib:int-add ix incx))
  (setf (f2cl-lib:fref x-%data%
                      (ix)
                      ((1 *))
                      x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
                     (ix)
                     ((1 *))
                     x-%offset%)
      (* temp
        (f2cl-lib:fref ap-%data%
                       (k)
                       ((1 *))
                       ap-%offset%))))))
(setf jx (f2cl-lib:int-add jx incx))
(setf kk
  (f2cl-lib:int-add kk
                    (f2cl-lib:int-add
                      (f2cl-lib:int-sub n j)
                      1))))))
(t
  (cond
    ((char-equal uplo #\U)
     (setf kk 1)
     (cond

```

```

(= incx 1)
(f2cl-lib:fdof (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf temp
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (setf k kk)
  (cond
    (noconj
      (f2cl-lib:fdof (i 1 (f2cl-lib:int-add i 1))
        (> i
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub
              1)))
          nil)
      (tagbody
        (setf temp
          (- temp
            (*
              (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%)
              (f2cl-lib:fref x-%data%
                (i)
                ((1 *))
                x-%offset%))))
          (setf k (f2cl-lib:int-add k 1))))
    (if nounit
      (setf temp
        (/ temp
          (f2cl-lib:fref ap-%data%
            ((f2cl-lib:int-sub
              (f2cl-lib:int-add kk j)
              1))
            ((1 *))
            ap-%offset%))))
      (t
        (f2cl-lib:fdof (i 1 (f2cl-lib:int-add i 1))
          (> i
            (f2cl-lib:int-add j
              (f2cl-lib:int-sub
                1)))
            nil)
          (tagbody
            (setf temp
              (- temp
                (*
                  (f2cl-lib:dconjg
                    (f2cl-lib:fref ap-%data%

```

```

(k)
((1 *))
ap-%offset%)
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%)))))
(setf k (f2cl-lib:int-add k 1)))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:dconjg
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk
              j)
            1))
          ((1 *))
          ap-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp)
    (setf kk (f2cl-lib:int-add kk j))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix kx)
    (cond
      (noconj
        (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
          (> k
            (f2cl-lib:int-add kk
              j
              (f2cl-lib:int-sub
                2)))
            nil)
        (tagbody
          (setf temp
            (- temp
              (*
                (f2cl-lib:fref ap-%data%
                  (k)
                  ((1 *))
                  ap-%offset%)
                (f2cl-lib:fref x-%data%
                  (ix)
                  ((1 *))

```

```

                                x-%offset%))))
    (setf ix (f2cl-lib:int-add ix incx)))
  (if nounit
    (setf temp
      (/ temp
        (f2cl-lib:fref ap-%data%
          ((f2cl-lib:int-sub
            (f2cl-lib:int-add kk j)
            1))
          ((1 *))
          ap-%offset%))))))
  (t
    (f2cl-lib:fdo (k kk (f2cl-lib:int-add k 1))
      (> k
        (f2cl-lib:int-add kk
          j
          (f2cl-lib:int-sub
            2)))
        nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref ap-%data%
                (k)
                ((1 *))
                ap-%offset%))
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))))
      (setf ix (f2cl-lib:int-add ix incx)))
    (if nounit
      (setf temp
        (/ temp
          (f2cl-lib:dconjg
            (f2cl-lib:fref ap-%data%
              ((f2cl-lib:int-sub
                (f2cl-lib:int-add kk
                  j)
                  1))
              ((1 *))
              ap-%offset%))))))
      (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
        temp)
      (setf jx (f2cl-lib:int-add jx incx))
      (setf kk (f2cl-lib:int-add kk j))))))
  (t
    (setf kk (the fixnum (truncate (* n (+ n 1) 2)))

```

```

(cond
  ((= incx 1)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      ((> j 1) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (setf k kk)
        (cond
          (noconj
            (f2cl-lib:fdo (i n
              (f2cl-lib:int-add i
                (f2cl-lib:int-sub 1)))
              ((> i (f2cl-lib:int-add j 1)) nil)
              (tagbody
                (setf temp
                  (- temp
                    (*
                     (f2cl-lib:fref ap-%data%
                       (k)
                       ((1 *))
                       ap-%offset%)
                     (f2cl-lib:fref x-%data%
                       (i)
                       ((1 *))
                       x-%offset%))))
                  (setf k (f2cl-lib:int-sub k 1))))
            (if nunit
              (setf temp
                (/ temp
                  (f2cl-lib:fref ap-%data%
                    ((f2cl-lib:int-add
                     (f2cl-lib:int-sub kk n)
                     j))
                    ((1 *))
                    ap-%offset%))))
              (t
                (f2cl-lib:fdo (i n
                  (f2cl-lib:int-add i
                    (f2cl-lib:int-sub 1)))
                    ((> i (f2cl-lib:int-add j 1)) nil)
                    (tagbody
                      (setf temp
                        (- temp
                          (*
                           (f2cl-lib:dconjg
                             (f2cl-lib:fref ap-%data%
                               (k)
                               ((1 *))
                               ap-%offset%))

```



```

nil)
(tagbody
  (setf temp
    (- temp
      (*
        (f2cl-lib:fref ap-%data%
          (k)
          ((1 *))
          ap-%offset%)
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))))
  (setf ix (f2cl-lib:int-sub ix incx)))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref ap-%data%
        ((f2cl-lib:int-add
          (f2cl-lib:int-sub kk n)
          j))
        ((1 *))
        ap-%offset%))))))
(t
  (f2cl-lib:fdo (k kk
    (f2cl-lib:int-add k
      (f2cl-lib:int-sub 1)))
    (> k
      (f2cl-lib:int-add kk
        (f2cl-lib:int-sub
          (f2cl-lib:int-add
            n
              (f2cl-lib:int-sub
                (f2cl-lib:int-add
                  j
                    1))))))
    nil)
  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref ap-%data%
              (k)
              ((1 *))
              ap-%offset%)
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))))

```

```

        (setf ix (f2cl-lib:int-sub ix incx)))
      (if nunit
        (setf temp
          (/ temp
             (f2cl-lib:dconjg
              (f2cl-lib:fref ap-%data%
                             ((f2cl-lib:int-add
                               (f2cl-lib:int-sub kk
                                                    n)
                               j))
                             ((1 *))
                             ap-%offset%))))))
        (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
              temp)
        (setf jx (f2cl-lib:int-sub jx incx))
        (setf kk
          (f2cl-lib:int-sub kk
                             (f2cl-lib:int-add
                              (f2cl-lib:int-sub n j)
                              1))))))
      end_label
      (return (values nil nil nil nil nil nil nil))))

```

ztrmv BLAS

— ztrmv.input —

```

)set break resume
)sys rm -f ztrmv.output
)spool ztrmv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztrmv.help —

```

=====
ztrmv examples

```

```
=====
=====
Man Page Details
=====
```

NAME

ZTRMV - perform one of the matrix-vector operations $x := A*x$, or $x := A'*x$, or $x := \text{conjg}(A')*x$,

SYNOPSIS

SUBROUTINE ZTRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

INTEGER INCX, LDA, N

CHARACTER*1 DIAG, TRANS, UPLO

COMPLEX*16 A(LDA, *), X(*)

PURPOSE

ZTRMV performs one of the matrix-vector operations

where x is an n element vector and A is an n by n unit, or non-unit, upper or lower triangular matrix.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $x := A*x$.

TRANS = 'T' or 't' $x := A'*x$.

TRANS = 'C' or 'c' $x := \text{conjg}(A')*x$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit

triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A. N must be at least zero. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, n). Unchanged on exit.

X - COMPLEX*16 array of dimension at least
(1 + (n - 1) * abs(INCX)). Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X. INCX must not be zero. Unchanged on exit.

— ztrmv.f —

SUBROUTINE ZTRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

```

*      .. Scalar Arguments ..
      INTEGER          INCX, LDA, N
      CHARACTER*1      DIAG, TRANS, UPLO
*      .. Array Arguments ..
      COMPLEX*16       A( LDA, * ), X( * )
*
*
*      Level 2 Blas routine.
*
*      -- Written on 22-October-1986.
*      Jack Dongarra, Argonne National Lab.
*      Jeremy Du Croz, Nag Central Office.
*      Sven Hammarling, Nag Central Office.
*      Richard Hanson, Sandia National Labs.
*
*
*      .. Parameters ..
      COMPLEX*16       ZERO
      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*      .. Local Scalars ..
      COMPLEX*16       TEMP
      INTEGER          I, INFO, IX, J, JX, KX
      LOGICAL          NOCONJ, NOUNIT
*      .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*      .. External Subroutines ..
      EXTERNAL         XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC        DCONJG, MAX
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ).AND.
$          .NOT.LSAME( UPLO , 'L' ) )THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ).AND.
$          .NOT.LSAME( TRANS, 'T' ).AND.
$          .NOT.LSAME( TRANS, 'C' ) )THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ).AND.
$          .NOT.LSAME( DIAG , 'N' ) )THEN
          INFO = 3
      ELSE IF( N.LT.0 )THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, N ) )THEN
          INFO = 6

```

```

ELSE IF( INCX.EQ.0 )THEN
    INFO = 8
END IF
IF( INFO.NE.0 )THEN
    CALL XERBLA( 'ZTRMV ', INFO )
    RETURN
END IF
*
* Quick return if possible.
*
IF( N.EQ.0 )
$ RETURN
*
NOCONJ = LSAME( TRANS, 'T' )
NOUNIT = LSAME( DIAG, 'N' )
*
* Set up the start point in X if the increment is not unity. This
* will be ( N - 1 ) * INCX too small for descending loops.
*
IF( INCX.LE.0 )THEN
    KX = 1 - ( N - 1 ) * INCX
ELSE IF( INCX.NE.1 )THEN
    KX = 1
END IF
*
* Start the operations. In this version the elements of A are
* accessed sequentially with one pass through A.
*
IF( LSAME( TRANS, 'N' ) )THEN
*
* Form x := A*x.
*
IF( LSAME( UPLO, 'U' ) )THEN
    IF( INCX.EQ.1 )THEN
        DO 20, J = 1, N
            IF( X( J ).NE.ZERO )THEN
                TEMP = X( J )
                DO 10, I = 1, J - 1
                    X( I ) = X( I ) + TEMP * A( I, J )
10                CONTINUE
                IF( NOUNIT )
$                    X( J ) = X( J ) * A( J, J )
                END IF
20            CONTINUE
        ELSE
            JX = KX
            DO 40, J = 1, N
                IF( X( JX ).NE.ZERO )THEN
                    TEMP = X( JX )
                    IX = KX

```

```

DO 30, I = 1, J - 1
    X( IX ) = X( IX ) + TEMP*A( I, J )
    IX      = IX      + INCX
30    CONTINUE
    IF( NOUNIT )
        $      X( JX ) = X( JX )*A( J, J )
    END IF
    JX = JX + INCX
40    CONTINUE
    END IF
ELSE
    IF( INCX.EQ.1 )THEN
        DO 60, J = N, 1, -1
            IF( X( J ).NE.ZERO )THEN
                TEMP = X( J )
                DO 50, I = N, J + 1, -1
                    X( I ) = X( I ) + TEMP*A( I, J )
50                CONTINUE
                IF( NOUNIT )
                    $      X( J ) = X( J )*A( J, J )
                END IF
60            CONTINUE
        ELSE
            KX = KX + ( N - 1 )*INCX
            JX = KX
            DO 80, J = N, 1, -1
                IF( X( JX ).NE.ZERO )THEN
                    TEMP = X( JX )
                    IX = KX
                    DO 70, I = N, J + 1, -1
                        X( IX ) = X( IX ) + TEMP*A( I, J )
                        IX      = IX      - INCX
70                    CONTINUE
                    IF( NOUNIT )
                        $      X( JX ) = X( JX )*A( J, J )
                    END IF
                    JX = JX - INCX
80                CONTINUE
            END IF
        END IF
    ELSE
        *
        *      Form x := A'*x or x := conjg( A' )*x.
        *
        IF( LSAME( UPLO, 'U' ) )THEN
            IF( INCX.EQ.1 )THEN
                DO 110, J = N, 1, -1
                    TEMP = X( J )
                    IF( NOCONJ )THEN
                        IF( NOUNIT )

```

[illegible]


```

        DO 160, I = J + 1, N
            TEMP = TEMP + DCONJG( A( I, J ) ) * X( I )
160      CONTINUE
        END IF
        X( J ) = TEMP
170      CONTINUE
    ELSE
        JX = KX
        DO 200, J = 1, N
            TEMP = X( JX )
            IX = JX
            IF( NOCONJ ) THEN
                IF( NOUNIT )
                    $      TEMP = TEMP * A( J, J )
                DO 180, I = J + 1, N
                    IX = IX + INCX
                    TEMP = TEMP + A( I, J ) * X( IX )
180              CONTINUE
                ELSE
                    IF( NOUNIT )
                        $      TEMP = TEMP * DCONJG( A( J, J ) )
                    DO 190, I = J + 1, N
                        IX = IX + INCX
                        TEMP = TEMP + DCONJG( A( I, J ) ) * X( IX )
190              CONTINUE
                    END IF
                    X( JX ) = TEMP
                    JX = JX + INCX
200          CONTINUE
        END IF
    END IF
END IF
*
RETURN
*
*   End of ZTRMV .
*
END

```

— BLAS 2 ztrmv —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztrmv (uplo trans diag n a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
              (type fixnum incx lda n)

```

```

        (type character diag trans uplo))
(f2cl-lib:with-multi-array-data
  ((uplo character uplo-%data% uplo-%offset%)
   (trans character trans-%data% trans-%offset%)
   (diag character diag-%data% diag-%offset%)
   (a (complex double-float) a-%data% a-%offset%)
   (x (complex double-float) x-%data% x-%offset%))
(prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
      (kx 0) (temp #C(0.0 0.0)))
(declare (type (member t nil) noconj nounit)
          (type fixnum i info ix j jx kx)
          (type (complex double-float) temp))
(setf info 0)
(cond
  ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
   (setf info 1))
  ((and (not (char-equal trans #\N))
        (not (char-equal trans #\T))
        (not (char-equal trans #\C)))
   (setf info 2))
  ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
   (setf info 3))
  (< n 0)
  (setf info 4))
(< lda (max (the fixnum 1) (the fixnum n)))
(setf info 6))
(= incx 0)
(setf info 8)))
(cond
  (/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "ZTRMV" info)
  (go end_label)))
(if (= n 0) (go end_label))
(setf noconj (char-equal trans #\T))
(setf nounit (char-equal diag #\N))
(cond
  (<= incx 0)
  (setf kx
    (f2cl-lib:int-sub 1
                      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                                         incx))))

  (/= incx 1)
  (setf kx 1)))
(cond
  (char-equal trans #\N)
  (cond
    (char-equal uplo #\U)
    (cond

```

```

(= incx 1)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref x (j) ((1 *))) zero)
      (setf temp
        (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub
              1)))
          nil)
      (tagbody
        (setf (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)
          (+
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))))))
      (if nunit
        (setf (f2cl-lib:fref x-%data%
          (j)
          ((1 *))
          x-%offset%)
          (*
            (f2cl-lib:fref x-%data%
              (j)
              ((1 *))
              x-%offset%)
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%))))))
      t
    (setf jx kx)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)

```

```

(setf temp
  (f2cl-lib:fref x-%data%
                (jx)
                ((1 *))
                x-%offset%))

(setf ix kx)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j
                      (f2cl-lib:int-sub
                      1)))
    nil)
(tagbody
  (setf (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%))
    (+
      (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))
  (setf ix (f2cl-lib:int-add ix incx)))
(if nount
  (setf (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
                    (jx)
                    ((1 *))
                    x-%offset%)
      (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%))))))
  (setf jx (f2cl-lib:int-add jx incx))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j 1) nil)
      (tagbody
        (cond

```

```

( (/= (f2cl-lib:fref x (j) ((1 *))) zero)
  (setf temp
    (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
  (f2cl-lib:fdo (i n
    (f2cl-lib:int-add i
      (f2cl-lib:int-sub 1)))
    ((> i (f2cl-lib:int-add j 1)) nil)
    (tagbody
      (setf (f2cl-lib:fref x-%data%
        (i)
        ((1 *))
        x-%offset%)
        (+
          (f2cl-lib:fref x-%data%
            (i)
            ((1 *))
            x-%offset%)
          (* temp
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%))))))
    (if nunit
      (setf (f2cl-lib:fref x-%data%
        (j)
        ((1 *))
        x-%offset%)
        (*
          (f2cl-lib:fref x-%data%
            (j)
            ((1 *))
            x-%offset%)
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))))
  (t
    (setf kx
      (f2cl-lib:int-add kx
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          incx)))
    (setf jx kx)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      ((> j 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
            (setf temp
              (f2cl-lib:fref x-%data%

```

```

                                (jx)
                                ((1 *))
                                x-%offset%)
(setf ix kx)
(f2cl-lib:fdo (i n
              (f2cl-lib:int-add i
                                (f2cl-lib:int-sub 1)))
              ((> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
      (+
        (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)
        (* temp
          (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%))))
      (setf ix (f2cl-lib:int-sub ix incx))))
(if nount
  (setf (f2cl-lib:fref x-%data%
                      (jx)
                      ((1 *))
                      x-%offset%)
    (*
      (f2cl-lib:fref x-%data%
                      (jx)
                      ((1 *))
                      x-%offset%)
      (f2cl-lib:fref a-%data%
                      (j j)
                      ((1 lda) (1 *))
                      a-%offset%))))))
  (setf jx (f2cl-lib:int-sub jx incx))))))
(t
  (cond
    ((char-equal uplo #\U)
      (cond
        ((= incx 1)
          (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                        ((> j 1) nil)
            (tagbody
              (setf temp
                (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
              (cond

```

```

(noconj
  (if nunit
    (setf temp
      (* temp
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%))))
    (f2cl-lib:fdo (i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub 1))
      (f2cl-lib:int-add i
        (f2cl-lib:int-sub 1)))
      (> i 1) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
  (t
    (if nunit
      (setf temp
        (* temp
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%))))
      (f2cl-lib:fdo (i
        (f2cl-lib:int-add j
          (f2cl-lib:int-sub 1))
        (f2cl-lib:int-add i
          (f2cl-lib:int-sub 1)))
        (> i 1) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:dconjg
                (f2cl-lib:fref a-%data%
                  (i j)
                  ((1 lda) (1 *))
                  a-%offset%))

```

```

(f2cl-lib:fref x-%data%
  (i)
  ((1 *))
  x-%offset%))))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%
      temp)))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (setf ix jx)
      (cond
        (noconj
          (if nounit
            (setf temp
              (* temp
                (f2cl-lib:fref a-%data%
                  (j j)
                  ((1 lda) (1 *))
                  a-%offset%)))
            (f2cl-lib:fdo (i
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub 1))
              (f2cl-lib:int-add i
                (f2cl-lib:int-sub 1)))
                (> i 1) nil)
              (tagbody
                (setf ix (f2cl-lib:int-sub ix incx))
                (setf temp
                  (+ temp
                    (*
                      (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)
                      (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%)))))))
          (t
            (if nounit
              (setf temp
                (* temp

```



```

(f2cl-lib:dconjg
  (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%))))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  (> i 1) nil)
(tagbody
  (setf ix (f2cl-lib:int-sub ix incx))
  (setf temp
    (+ temp
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (i j)
            ((1 lda) (1 *))
            a-%offset%))
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))))
  (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
    temp)
  (setf jx (f2cl-lib:int-sub jx incx))))))
(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (setf temp
          (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
        (cond
          (noconj
            (if nunit
              (setf temp
                (* temp
                  (f2cl-lib:fref a-%data%
                    (j j)
                    ((1 lda) (1 *))
                    a-%offset%))))
            (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
              (f2cl-lib:int-add i 1))
                (> i n) nil)
              (tagbody
                (setf temp

```

```

(+ temp
  (*
    (f2cl-lib:fref a-%data%
      (i j)
      ((1 lda) (1 *))
      a-%offset%)
    (f2cl-lib:fref x-%data%
      (i)
      ((1 *))
      x-%offset%))))))
(t
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
        (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
          temp))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
    (setf ix jx)
    (cond
      (noconj
        (if nounit
          (setf temp
            (* temp

```

```

(f2cl-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf ix (f2cl-lib:int-add ix incx))
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))))
(t
  (if nunit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))))
  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
    (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf ix (f2cl-lib:int-add ix incx))
    (setf temp
      (+ temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%))
          (f2cl-lib:fref x-%data%
            (ix)
            ((1 *))
            x-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
      temp)
    (setf jx (f2cl-lib:int-add jx incx))))))

```

end_label

```
(return (values nil nil nil nil nil nil nil nil nil))))))
```

ztrsv BLAS

— ztrsv.input —

```
)set break resume
)sys rm -f ztrsv.output
)spool ztrsv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— ztrsv.help —

```
=====
ztrsv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZTRSV - solve one of the systems of equations $A*x = b$, or $A'*x = b$, or $\text{conjg}(A')*x = b$,

SYNOPSIS

SUBROUTINE ZTRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)

INTEGER INCX, LDA, N

CHARACTER*1 DIAG, TRANS, UPLO

COMPLEX*16 A(LDA, *), X(*)

PURPOSE

ZTRSV solves one of the systems of equations

where b and x are n element vectors and A is an n by n unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n' $Ax = b$.

TRANS = 'T' or 't' $A^T x = b$.

TRANS = 'C' or 'c' $\text{conjg}(A)x = b$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix A . N must be at least zero. Unchanged on exit.

A - COMPLEX*16 array of DIMENSION (LDA, n).

Before entry with UPLO = 'U' or 'u', the leading n

by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with $UPLO = 'L'$ or $'l'$, the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when $DIAG = 'U'$ or $'u'$, the diagonal elements of A are not referenced either, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least $\max(1, n)$. Unchanged on exit.

X - COMPLEX*16 array of dimension at least $(1 + (n - 1) * \text{abs}(INCX))$. Before entry, the incremented array X must contain the n element right-hand side vector b . On exit, X is overwritten with the solution vector x .

INCX - INTEGER.

On entry, INCX specifies the increment for the elements of X . INCX must not be zero. Unchanged on exit.

— ztrsv.f —

```

SUBROUTINE ZTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
* .. Scalar Arguments ..
  INTEGER          INCX, LDA, N
  CHARACTER*1      DIAG, TRANS, UPLO
* .. Array Arguments ..
  COMPLEX*16       A( LDA, * ), X( * )
* ..
*
* Level 2 Blas routine.
*
* -- Written on 22-October-1986.
*   Jack Dongarra, Argonne National Lab.
*   Jeremy Du Croz, Nag Central Office.
*   Sven Hammarling, Nag Central Office.
*   Richard Hanson, Sandia National Labs.
*
*

```

```

*      .. Parameters ..
      COMPLEX*16      ZERO
      PARAMETER      ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*      .. Local Scalars ..
      COMPLEX*16      TEMP
      INTEGER          I, INFO, IX, J, JX, KX
      LOGICAL          NOCONJ, NOUNIT
*      .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*      .. External Subroutines ..
      EXTERNAL         XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC        DCONJG, MAX
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF      ( .NOT.LSAME( UPLO , 'U' ) .AND.
$           .NOT.LSAME( UPLO , 'L' ) ) THEN
          INFO = 1
      ELSE IF( .NOT.LSAME( TRANS, 'N' ) .AND.
$           .NOT.LSAME( TRANS, 'T' ) .AND.
$           .NOT.LSAME( TRANS, 'C' ) ) THEN
          INFO = 2
      ELSE IF( .NOT.LSAME( DIAG , 'U' ) .AND.
$           .NOT.LSAME( DIAG , 'N' ) ) THEN
          INFO = 3
      ELSE IF( N.LT.0 ) THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = 6
      ELSE IF( INCX.EQ.0 ) THEN
          INFO = 8
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZTRSV ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
      NOCONJ = LSAME( TRANS, 'T' )
      NOUNIT = LSAME( DIAG , 'N' )
*

```

```

*      Set up the start point in X if the increment is not unity. This
*      will be ( N - 1 ) * INCX too small for descending loops.
*
      IF( INCX.LE.0 )THEN
        KX = 1 - ( N - 1 ) * INCX
      ELSE IF( INCX.NE.1 )THEN
        KX = 1
      END IF

*
*      Start the operations. In this version the elements of A are
*      accessed sequentially with one pass through A.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*        Form x := inv( A ) * x.
*
        IF( LSAME( UPLO, 'U' ) )THEN
          IF( INCX.EQ.1 )THEN
            DO 20, J = N, 1, -1
              IF( X( J ).NE.ZERO )THEN
                IF( NUNIT )
$                  X( J ) = X( J ) / A( J, J )
                TEMP = X( J )
                DO 10, I = J - 1, 1, -1
                  X( I ) = X( I ) - TEMP * A( I, J )
10                CONTINUE
              END IF
            CONTINUE
          ELSE
            JX = KX + ( N - 1 ) * INCX
            DO 40, J = N, 1, -1
              IF( X( JX ).NE.ZERO )THEN
                IF( NUNIT )
$                  X( JX ) = X( JX ) / A( J, J )
                TEMP = X( JX )
                IX = JX
                DO 30, I = J - 1, 1, -1
                  IX = IX - INCX
                  X( IX ) = X( IX ) - TEMP * A( I, J )
30                CONTINUE
              END IF
              JX = JX - INCX
            CONTINUE
          END IF
        ELSE
          IF( INCX.EQ.1 )THEN
            DO 60, J = 1, N
              IF( X( J ).NE.ZERO )THEN
                IF( NUNIT )
$                  X( J ) = X( J ) / A( J, J )

```



```

        TEMP = X( J )
        DO 50, I = J + 1, N
            X( I ) = X( I ) - TEMP*A( I, J )
50      CONTINUE
        END IF
60      CONTINUE
    ELSE
        JX = KX
        DO 80, J = 1, N
            IF( X( JX ).NE.ZERO )THEN
                IF( NOUNIT )
                    $      X( JX ) = X( JX )/A( J, J )
                TEMP = X( JX )
                IX = JX
                DO 70, I = J + 1, N
                    IX = IX + INCX
                    X( IX ) = X( IX ) - TEMP*A( I, J )
70              CONTINUE
                END IF
                JX = JX + INCX
80          CONTINUE
            END IF
        END IF
    ELSE
*
*      Form x := inv( A' )*x or x := inv( conjg( A' ) )*x.
*
        IF( LSAME( UPLO, 'U' ) )THEN
            IF( INCX.EQ.1 )THEN
                DO 110, J = 1, N
                    TEMP = X( J )
                    IF( NOCONJ )THEN
                        DO 90, I = 1, J - 1
                            TEMP = TEMP - A( I, J )*X( I )
90              CONTINUE
                        IF( NOUNIT )
                            $      TEMP = TEMP/A( J, J )
                    ELSE
                        DO 100, I = 1, J - 1
                            TEMP = TEMP - DCONJG( A( I, J ) )*X( I )
100             CONTINUE
                        IF( NOUNIT )
                            $      TEMP = TEMP/DCONJG( A( J, J ) )
                        END IF
                        X( J ) = TEMP
110          CONTINUE
            ELSE
                JX = KX
                DO 140, J = 1, N
                    IX = KX

```

```

TEMP = X( JX )
IF( NOCONJ )THEN
    DO 120, I = 1, J - 1
        TEMP = TEMP - A( I, J ) * X( IX )
        IX = IX + INCX
120    CONTINUE
    IF( NUNIT )
        $    TEMP = TEMP / A( J, J )
    ELSE
        DO 130, I = 1, J - 1
            TEMP = TEMP - DCONJG( A( I, J ) ) * X( IX )
            IX = IX + INCX
130    CONTINUE
        IF( NUNIT )
            $    TEMP = TEMP / DCONJG( A( J, J ) )
        END IF
        X( JX ) = TEMP
        JX = JX + INCX
140    CONTINUE
    END IF
ELSE
    IF( INCX.EQ.1 )THEN
        DO 170, J = N, 1, -1
            TEMP = X( J )
            IF( NOCONJ )THEN
                DO 150, I = N, J + 1, -1
                    TEMP = TEMP - A( I, J ) * X( I )
150                CONTINUE
                IF( NUNIT )
                    $    TEMP = TEMP / A( J, J )
                ELSE
                    DO 160, I = N, J + 1, -1
                        TEMP = TEMP - DCONJG( A( I, J ) ) * X( I )
160                    CONTINUE
                    IF( NUNIT )
                        $    TEMP = TEMP / DCONJG( A( J, J ) )
                    END IF
                    X( J ) = TEMP
170                CONTINUE
            ELSE
                KX = KX + ( N - 1 ) * INCX
                JX = KX
                DO 200, J = N, 1, -1
                    IX = KX
                    TEMP = X( JX )
                    IF( NOCONJ )THEN
                        DO 180, I = N, J + 1, -1
                            TEMP = TEMP - A( I, J ) * X( IX )
                            IX = IX - INCX
180                        CONTINUE

```

```

                IF( NOUNIT )
$                TEMP = TEMP/A( J, J )
                ELSE
                DO 190, I = N, J + 1, -1
                    TEMP = TEMP - DCONJG( A( I, J ) ) * X( IX )
                    IX = IX - INCX
190                CONTINUE
                IF( NOUNIT )
$                TEMP = TEMP/DCONJG( A( J, J ) )
                END IF
                X( JX ) = TEMP
                JX = JX - INCX
200            CONTINUE
            END IF
        END IF
    END IF
*
*   RETURN
*
*   End of ZTRSV .
*
*   END

```

— BLAS 2 ztrsv —

```

(let* ((zero (complex 0.0 0.0)))
  (declare (type (complex double-float) zero))
  (defun ztrsv (uplo trans diag n a lda x incx)
    (declare (type (simple-array (complex double-float) (*)) x a)
              (type fixnum incx lda n)
              (type character diag trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (x (complex double-float) x-%data% x-%offset%))
      (prog ((noconj nil) (nounit nil) (i 0) (info 0) (ix 0) (j 0) (jx 0)
             (kx 0) (temp #C(0.0 0.0)))
        (declare (type (member t nil) noconj nounit)
                  (type fixnum i info ix j jx kx)
                  (type (complex double-float) temp))
        (setf info 0)
        (cond
          ((and (not (char-equal uplo #\U)) (not (char-equal uplo #\L)))
           (setf info 1))

```

```

((and (not (char-equal trans #\N))
      (not (char-equal trans #\T))
      (not (char-equal trans #\C)))
 (setf info 2))
((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
 (setf info 3))
(< n 0)
 (setf info 4))
(< lda (max (the fixnum 1) (the fixnum n)))
 (setf info 6))
(= incx 0)
 (setf info 8)))
(cond
  (/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "ZTRSV" info)
  (go end_label)))
(if (= n 0) (go end_label))
(setf noconj (char-equal trans #\T))
(setf nounit (char-equal diag #\N))
(cond
  (<= incx 0)
  (setf kx
    (f2cl-lib:int-sub 1
      (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
        incx))))

  (/= incx 1)
  (setf kx 1)))
(cond
  ((char-equal trans #\N)
   (cond
     ((char-equal uplo #\U)
      (cond
        (= incx 1)
        (f2cl-lib:fd0 (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
          (> j 1) nil)

        (tagbody
          (cond
            (/= (f2cl-lib:fref x (j) ((1 *))) zero)
            (if nounit
              (setf (f2cl-lib:fref x-%data%
                                   (j)
                                   ((1 *))
                                   x-%offset%)
                (/
                  (f2cl-lib:fref x-%data%
                                   (j)
                                   ((1 *))
                                   x-%offset%)

```

```

(f2cl-lib:fref a-%data%
  (j j)
  ((1 lda) (1 *))
  a-%offset%)))
(setf temp
  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  ((> i 1) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
      (i)
      ((1 *))
      x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)
        (* temp
          (f2cl-lib:fref a-%data%
            (i j)
            ((1 lda) (1 *))
            a-%offset%))))))
(t
  (setf jx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j 1) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (if nunit
            (setf (f2cl-lib:fref x-%data%
              (jx)
              ((1 *))
              x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                  (jx)
                  ((1 *))
                  x-%offset%)
                (f2cl-lib:fref a-%data%
                  (j j)

```

```

((1 lda) (1 *))
a-%offset%)))

(setf temp
  (f2cl-lib:fref x-%data%
    (jx)
    ((1 *))
    x-%offset%))

(setf ix jx)
(f2cl-lib:fdo (i
  (f2cl-lib:int-add j
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add i
    (f2cl-lib:int-sub 1)))
  (> i 1) nil)

(tagbody
  (setf ix (f2cl-lib:int-sub ix incx))
  (setf (f2cl-lib:fref x-%data%
    (ix)
    ((1 *))
    x-%offset%)
    (-
      (f2cl-lib:fref x-%data%
        (ix)
        ((1 *))
        x-%offset%)
      (* temp
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%))))))

(setf jx (f2cl-lib:int-sub jx incx))))))

(t
  (cond
    ((= incx 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref x (j) ((1 *)) zero)
            (if nounit
              (setf (f2cl-lib:fref x-%data%
                (j)
                ((1 *))
                x-%offset%)
                (/
                  (f2cl-lib:fref x-%data%
                    (j)
                    ((1 *))
                    x-%offset%)
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%))))))
            (setf jx (f2cl-lib:int-sub jx incx))))))
    (t
      (setf jx (f2cl-lib:int-sub jx incx))))))

```

```

                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%)))
(setf temp
  (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
              (f2cl-lib:int-add i 1))
              (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
                        (i)
                        ((1 *))
                        x-%offset%)
        (* temp
          (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                (> j n) nil)
    (tagbody
      (cond
        ((/= (f2cl-lib:fref x (jx) ((1 *))) zero)
          (if nunit
            (setf (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
              (/
                (f2cl-lib:fref x-%data%
                                (jx)
                                ((1 *))
                                x-%offset%)
                (f2cl-lib:fref a-%data%
                                (j j)
                                ((1 lda) (1 *))
                                a-%offset%))))
            (setf temp
              (f2cl-lib:fref x-%data%
                              (jx)
                              ((1 *))
                              x-%offset%))
            (setf ix jx)
          )
        )
      )

```

```
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
               (f2cl-lib:int-add i 1))
              (> i n) nil)

(tagbody
 (setf ix (f2cl-lib:int-add ix incx))
 (setf (f2cl-lib:fref x-%data%
                     (ix)
                     ((1 *))
                     x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
                      (ix)
                      ((1 *))
                      x-%offset%)
        (* temp
          (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)))))))
(setf jx (f2cl-lib:int-add jx incx)))))))))

(t
 (cond
  ((char-equal uplo #\U)
   (cond
    ((= incx 1)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   (> j n) nil)
     (tagbody
      (setf temp
              (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
      (cond
       (noconj
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      (> i
                       (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                 1)))
                      nil)
        (tagbody
         (setf temp
                 (- temp
                    (*
                     (f2cl-lib:fref a-%data%
                                     (i j)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                     (f2cl-lib:fref x-%data%
                                     (i)
                                     ((1 *))
                                     x-%offset%))))))
```



```

(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%))))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub
          1)))
      nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))))
    (if nunit
      (setf temp
        (/ temp
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%)
      temp)))
(t
  (setf jx kx)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf ix kx)
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (cond
        (noconj
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i
              (f2cl-lib:int-add j

```

```

(f2cl-lib:int-sub
1)))

nil)

(tagbody
  (setf temp
    (- temp
      (*
        (f2cl-lib:fref a-%data%
          (i j)
          ((1 lda) (1 *))
          a-%offset%))
        (f2cl-lib:fref x-%data%
          (ix)
          ((1 *))
          x-%offset%))))))
  (setf ix (f2cl-lib:int-add ix incx))))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
        (j j)
        ((1 lda) (1 *))
        a-%offset%))))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub
          1)))

    nil)

  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%))
            (f2cl-lib:fref x-%data%
              (ix)
              ((1 *))
              x-%offset%))))))
      (setf ix (f2cl-lib:int-add ix incx))))
  (if nunit
    (setf temp
      (/ temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (j j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (f2cl-lib:fref x-%data%
      (ix)
      ((1 *))
      x-%offset%))))))
  (setf ix (f2cl-lib:int-add ix incx))))

```

```

((1 lda) (1 *))
a-%offset%))))))
(setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
temp)
(setf jx (f2cl-lib:int-add jx incx))))))
(t
(cond
(= incx 1)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
(> j 1) nil)
(tagbody
(setf temp
(f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%))
(cond
(noconj
(f2cl-lib:fdo (i n
(f2cl-lib:int-add i
(f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1)) nil)
(tagbody
(setf temp
(- temp
(*
(f2cl-lib:fref a-%data%
(i j)
((1 lda) (1 *))
a-%offset%)
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%))))))
(if nunit
(setf temp
(/ temp
(f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%))))))
(t
(f2cl-lib:fdo (i n
(f2cl-lib:int-add i
(f2cl-lib:int-sub 1)))
(> i (f2cl-lib:int-add j 1)) nil)
(tagbody
(setf temp
(- temp
(*
(f2cl-lib:dconjg
(f2cl-lib:fref a-%data%
(i j)

```

```

((1 lda) (1 *))
a-%offset%))
(f2cl-lib:fref x-%data%
(i)
((1 *))
x-%offset%))))))
(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:dconjg
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%))))))
(setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%
temp))))
(t
  (setf kx
    (f2cl-lib:int-add kx
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub n 1)
        incx)))
    (setf jx kx)
    (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
      (> j 1) nil)
    (tagbody
      (setf ix kx)
      (setf temp
        (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%))
      (cond
        (noconj
          (f2cl-lib:fdo (i n
            (f2cl-lib:int-add i
              (f2cl-lib:int-sub 1)))
            (> i (f2cl-lib:int-add j 1)) nil)
          (tagbody
            (setf temp
              (- temp
                (*
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%)
                  (f2cl-lib:fref x-%data%
                    (ix)
                    ((1 *))
                    x-%offset%))))
              (setf ix (f2cl-lib:int-sub ix incx))))
            (if nunit
              (setf temp

```

```

        (/ temp
          (f2cl-lib:fref a-%data%
                        (j j)
                        ((1 lda) (1 *))
                        a-%offset%))))))
(t
  (f2cl-lib:fdo (i n
                (f2cl-lib:int-add i
                                (f2cl-lib:int-sub 1)))
    ((> i (f2cl-lib:int-add j 1)) nil)
  (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%))
          (f2cl-lib:fref x-%data%
                        (ix)
                        ((1 *))
                        x-%offset%))))))
    (setf ix (f2cl-lib:int-sub ix incx)))
  (if nunit
    (setf temp
      (/ temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
                        (j j)
                        ((1 lda) (1 *))
                        a-%offset%))))))
    (setf (f2cl-lib:fref x-%data% (jx) ((1 *)) x-%offset%)
      temp)
    (setf jx (f2cl-lib:int-sub jx incx))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

Chapter 5

BLAS Level 3

dgemm BLAS

— dgemm.input —

```
)set break resume
)sys rm -f dgemm.output
)spool dgemm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

—————
— dgemm.help —

```
=====
dgemm examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGEMM - perform one of the matrix-matrix operations C :=
alpha*op(A)*op(B) + beta*C,

SYNOPSIS

```
SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
                  B, LDB, BETA, C, LDC )
```

```
CHARACTER*1  TRANSA, TRANSB
```

```
INTEGER      M, N, K, LDA, LDB, LDC
```

```
DOUBLE       PRECISION ALPHA, BETA
```

```
DOUBLE       PRECISION A( LDA, * ), B( LDB, * ), C( LDC,
                  * )
```

PURPOSE

DGEMM performs one of the matrix-matrix operations

where $\text{op}(X)$ is one of

$$\text{op}(X) = X \quad \text{or} \quad \text{op}(X) = X',$$

alpha and beta are scalars, and A, B and C are matrices,
with $\text{op}(A)$ an m by k matrix, $\text{op}(B)$ a k by n matrix
and C an m by n matrix.

PARAMETERS

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n', $\text{op}(A) = A$.

TRANSA = 'T' or 't', $\text{op}(A) = A'$.

TRANSA = 'C' or 'c', $\text{op}(A) = A'$.

Unchanged on exit.

TRANSB - CHARACTER*1. On entry, TRANSB specifies the form of $\text{op}(B)$ to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', $\text{op}(B) = B$.

TRANSB = 'T' or 't', $\text{op}(B) = B'$.

TRANSB = 'C' or 'c', $\text{op}(B) = B'$.

Unchanged on exit.

M - INTEGER.

On entry, *M* specifies the number of rows of the matrix *op*(*A*) and of the matrix *C*. *M* must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, *N* specifies the number of columns of the matrix *op*(*B*) and the number of columns of the matrix *C*. *N* must be at least zero. Unchanged on exit.

K - INTEGER.

On entry, *K* specifies the number of columns of the matrix *op*(*A*) and the number of rows of the matrix *op*(*B*). *K* must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A -

DOUBLE PRECISION array of DIMENSION (*LDA*, *ka*), where *k* when *TRANSA* = 'N' or 'n', and is *m* otherwise. Before entry with *TRANSA* = 'N' or 'n', the leading *m* by *k* part of the array *A* must contain the matrix *A*, otherwise the leading *k* by *m* part of the array *A* must contain the matrix *A*. Unchanged on exit.

LDA - INTEGER.

On entry, *LDA* specifies the first dimension of *A* as declared in the calling (sub) program. When *TRANSA* = 'N' or 'n' then *LDA* must be at least $\max(1, m)$, otherwise *LDA* must be at least $\max(1, k)$. Unchanged on exit.

kb is

B -

DOUBLE PRECISION array of DIMENSION (*LDB*, *kb*), where *n* when *TRANSB* = 'N' or 'n', and is *k* otherwise. Before entry with *TRANSB* = 'N' or 'n', the leading *k* by *n* part of the array *B* must contain the matrix *B*, otherwise the leading *n* by *k* part of the array *B* must contain the matrix *B*. Unchanged on exit.

LDB - INTEGER.

On entry, *LDB* specifies the first dimension of *B* as declared in the calling (sub) program. When *TRANSB* = 'N' or 'n' then *LDB* must be at least $\max(1, k)$,

otherwise LDB must be at least $\max(1, n)$.
 Unchanged on exit.

BETA - DOUBLE PRECISION.
 On entry, BETA specifies the scalar beta. When
 BETA is supplied as zero then C need not be set on
 input. Unchanged on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n).
 Before entry, the leading m by n part of the array
 C must contain the matrix C, except when beta is
 zero, in which case C need not be set on entry. On
 exit, the array C is overwritten by the m by n
 matrix $(\alpha * \text{op}(A) * \text{op}(B) + \beta * C)$.

LDC - INTEGER.
 On entry, LDC specifies the first dimension of C as
 declared in the calling (sub) program. LDC
 must be at least $\max(1, m)$. Unchanged on exit.

—————

— dgemm.f —

```

SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
$                BETA, C, LDC )
*   .. Scalar Arguments ..
CHARACTER*1      TRANSA, TRANSB
INTEGER          M, N, K, LDA, LDB, LDC
DOUBLE PRECISION ALPHA, BETA
*   .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), B( LDB, * ), C( LDC, * )
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*   .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
*   .. External Subroutines ..
EXTERNAL         XERBLA

```

```

*      .. Intrinsic Functions ..
      INTRINSIC          MAX
*      .. Local Scalars ..
      LOGICAL            NOTA, NOTB
      INTEGER            I, INFO, J, L, NCOLA, NROWA, NROWB
      DOUBLE PRECISION   TEMP
*      .. Parameters ..
      DOUBLE PRECISION   ONE          , ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      .. Executable Statements ..
*
*      Set NOTA and NOTB as true if A and B respectively are not
*      transposed and set NROWA, NCOLA and NROWB as the number of rows
*      and columns of A and the number of rows of B respectively.
*
      NOTA = LSAME( TRANSA, 'N' )
      NOTB = LSAME( TRANSB, 'N' )
      IF( NOTA )THEN
        NROWA = M
        NCOLA = K
      ELSE
        NROWA = K
        NCOLA = M
      END IF
      IF( NOTB )THEN
        NROWB = K
      ELSE
        NROWB = N
      END IF
*
*      Test the input parameters.
*
      INFO = 0
      IF( ( .NOT.NOTA ) .AND.
$      ( .NOT.LSAME( TRANSA, 'C' ) ) .AND.
$      ( .NOT.LSAME( TRANSA, 'T' ) ) ) THEN
        INFO = 1
      ELSE IF( ( .NOT.NOTB ) .AND.
$      ( .NOT.LSAME( TRANSB, 'C' ) ) .AND.
$      ( .NOT.LSAME( TRANSB, 'T' ) ) ) THEN
        INFO = 2
      ELSE IF( M .LT.0 ) THEN
        INFO = 3
      ELSE IF( N .LT.0 ) THEN
        INFO = 4
      ELSE IF( K .LT.0 ) THEN
        INFO = 5
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
        INFO = 8

```

```

ELSE IF( LDB.LT.MAX( 1, NROWB ) )THEN
    INFO = 10
ELSE IF( LDC.LT.MAX( 1, M      ) )THEN
    INFO = 13
END IF
IF( INFO.NE.0 )THEN
    CALL XERBLA( 'DGEMM ', INFO )
    RETURN
END IF

*
* Quick return if possible.
*
IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$   ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
$   RETURN

*
* And if alpha.eq.zero.
*
IF( ALPHA.EQ.ZERO )THEN
    IF( BETA.EQ.ZERO )THEN
        DO 20, J = 1, N
            DO 10, I = 1, M
                C( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
    ELSE
        DO 40, J = 1, N
            DO 30, I = 1, M
                C( I, J ) = BETA*C( I, J )
30          CONTINUE
40          CONTINUE
    END IF
    RETURN
END IF

*
* Start the operations.
*
IF( NOTB )THEN
    IF( NOTA )THEN

*
* Form C := alpha*A*B + beta*C.
*
        DO 90, J = 1, N
            IF( BETA.EQ.ZERO )THEN
                DO 50, I = 1, M
                    C( I, J ) = ZERO
50              CONTINUE
            ELSE IF( BETA.NE.ONE )THEN
                DO 60, I = 1, M
                    C( I, J ) = BETA*C( I, J )

```

```

60          CONTINUE
          END IF
          DO 80, L = 1, K
            IF( B( L, J ).NE.ZERO )THEN
              TEMP = ALPHA*B( L, J )
              DO 70, I = 1, M
                C( I, J ) = C( I, J ) + TEMP*A( I, L )
70          CONTINUE
            END IF
          CONTINUE
80          CONTINUE
90          CONTINUE
        ELSE
*
*          Form C := alpha*A'*B + beta*C
*
          DO 120, J = 1, N
            DO 110, I = 1, M
              TEMP = ZERO
              DO 100, L = 1, K
                TEMP = TEMP + A( L, I )*B( L, J )
100          CONTINUE
              IF( BETA.EQ.ZERO )THEN
                C( I, J ) = ALPHA*TEMP
              ELSE
                C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
              END IF
110          CONTINUE
120          CONTINUE
            END IF
          ELSE
            IF( NOTA )THEN
*
*          Form C := alpha*A*B' + beta*C
*
              DO 170, J = 1, N
                IF( BETA.EQ.ZERO )THEN
                  DO 130, I = 1, M
                    C( I, J ) = ZERO
130          CONTINUE
                ELSE IF( BETA.NE.ONE )THEN
                  DO 140, I = 1, M
                    C( I, J ) = BETA*C( I, J )
140          CONTINUE
                END IF
              DO 160, L = 1, K
                IF( B( J, L ).NE.ZERO )THEN
                  TEMP = ALPHA*B( J, L )
                  DO 150, I = 1, M
                    C( I, J ) = C( I, J ) + TEMP*A( I, L )
150          CONTINUE

```

```

                                END IF
160                                CONTINUE
170                                CONTINUE
                                ELSE
*
*                                Form C := alpha*A'*B' + beta*C
*
                                DO 200, J = 1, N
                                    DO 190, I = 1, M
                                        TEMP = ZERO
                                        DO 180, L = 1, K
                                            TEMP = TEMP + A( L, I ) * B( J, L )
180                                        CONTINUE
                                        IF( BETA.EQ.ZERO ) THEN
                                            C( I, J ) = ALPHA*TEMP
                                        ELSE
                                            C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                                        END IF
190                                    CONTINUE
200                                CONTINUE
                                END IF
                                END IF
*
*                                RETURN
*
*                                End of DGEMM .
*
                                END

```

— BLAS 3 dgemm —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dgemm (transa transb m n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
              (type (double-float) beta alpha)
              (type fixnum ldc ldb$ lda k n m)
              (type character transb transa))
    (f2cl-lib:with-multi-array-data
      ((transa character transa-%data% transa-%offset%)
       (transb character transb-%data% transb-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%)
       (c double-float c-%data% c-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (l 0) (ncola 0) (nrowa 0)

```

```

        (nrowb 0) (nota nil) (notb nil))
(declare (type (double-float) temp)
          (type fixnum i info j l ncola nrowa nrowb)
          (type (member t nil) nota notb))
(setf nota (char-equal transa #\N))
(setf notb (char-equal transb #\N))
(cond
  (nota
   (setf nrowa m)
   (setf ncola k))
  (t
   (setf nrowa k)
   (setf ncola m)))
(cond
  (notb
   (setf nrowb k))
  (t
   (setf nrowb n)))
(setf info 0)
(cond
  ((and (not nota) (not (char-equal transa #\C))
        (not (char-equal transa #\T))))
   (setf info 1))
  ((and (not notb) (not (char-equal transb #\C))
        (not (char-equal transb #\T))))
   (setf info 2))
  ((< m 0)
   (setf info 3))
  ((< n 0)
   (setf info 4))
  ((< k 0)
   (setf info 5))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 8))
  ((< ldb$
    (max (the fixnum 1) (the fixnum nrowb)))
   (setf info 10))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond

```

```

(= beta zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
        zero))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
            (* beta
              (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%))))))))))
  (go end_label)))
(cond
  (notb
    (cond
      (nota
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
          (tagbody
            (cond
              ((= beta zero)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  (> i m) nil)
                  (tagbody
                    (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%)
                      zero))))))
              ((/= beta one)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  (> i m) nil)
                  (tagbody

```



```

      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%))
      (* beta
        (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref b (l j) ((1 ldb$) (1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref b-%data%
                          (l j)
                          ((1 ldb$) (1 *))
                          b-%offset%)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%))
            (+
              (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              (* temp
                (f2cl-lib:fref a-%data%
                                (i l)
                                ((1 lda) (1 *))
                                a-%offset%))))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf temp zero)
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
        (> 1 k) nil)
      (tagbody
        (setf temp
          (+ temp
            (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                              (i l)
                              ((1 lda) (1 *))
                              a-%offset%))))))))))

```

```

(*
  (f2cl-lib:fref a-%data%
                (1 i)
                ((1 lda) (1 *))
                a-%offset%)
  (f2cl-lib:fref b-%data%
                (1 j)
                ((1 ldb$) (1 *))
                b-%offset%))))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
         (* alpha temp)))
  (t
   (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
         (+ (* alpha temp)
            (* beta
              (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%))))))))))
(t
  (cond
    (nota
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
         (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                zero))))
        ((/= beta one)
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                       ((> i m) nil)
         (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))

```

```

                                c-%offset%)
      (* beta
        (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *))
                      c-%offset%))))))
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero)
      (setf temp
        (* alpha
          (f2cl-lib:fref b-%data%
                        (j 1)
                        ((1 ldb$) (1 *))
                        b-%offset%)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
            (+
              (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
              (* temp
                (f2cl-lib:fref a-%data%
                              (i 1)
                              ((1 lda) (1 *))
                              a-%offset%))))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf temp zero)
      (f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
        (> 1 k) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:fref a-%data%
                            (1 i)

```

```

                                ((1 lda) (1 *))
                                a-%offset%)
(f2cl-lib:fref b-%data%
  (j 1)
  ((1 ldb$) (1 *))
  b-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* alpha temp)))
  (t
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (+ (* alpha temp)
             (* beta
               (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%))))))))))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil nil)))

```

dsymm BLAS

— dsymm.input —

```

)set break resume
)sys rm -f dsymm.output
)spool dsymm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dsymm.help —

```
=====
dsymm examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DSYMM - perform one of the matrix-matrix operations $C := \alpha A * B + \beta C$,

SYNOPSIS

```
SUBROUTINE DSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
                  BETA, C, LDC )
```

```
CHARACTER*1 SIDE, UPLO
```

```
INTEGER      M, N, LDA, LDB, LDC
```

```
DOUBLE       PRECISION ALPHA, BETA
```

```
DOUBLE       PRECISION A( LDA, * ), B( LDB, * ), C( LDC,
* )
```

PURPOSE

DSYMM performs one of the matrix-matrix operations

or

```
C := alpha*B*A + beta*C,
```

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

```
SIDE = 'L' or 'l'  C := alpha*A*B + beta*C,
```

```
SIDE = 'R' or 'r'  C := alpha*B*A + beta*C,
```

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix C. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix C. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

-
DOUBLE PRECISION array of DIMENSION (LDA, ka), where m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part

of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least $\max(1, m)$, otherwise LDA must be at least $\max(1, n)$. Unchanged on exit.

B - DOUBLE PRECISION array of DIMENSION (LDB, n).

Before entry, the leading m by n part of the array B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least $\max(1, m)$. Unchanged on exit.

BETA - DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n).

Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC - INTEGER.

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, m)$. Unchanged on exit.

Level 3 Blas routine.

— dsymm.f —

```

SUBROUTINE DSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
$               BETA, C, LDC )
* .. Scalar Arguments ..
CHARACTER*1      SIDE, UPLO

```

```

      INTEGER          M, N, LDA, LDB, LDC
      DOUBLE PRECISION ALPHA, BETA
*    .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), B( LDB, * ), C( LDC, * )
*
*    ..
*
*    Level 3 Blas routine.
*
*    -- Written on 8-February-1989.
*       Jack Dongarra, Argonne National Laboratory.
*       Iain Duff, AERE Harwell.
*       Jeremy Du Croz, Numerical Algorithms Group Ltd.
*       Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*    .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL          LSAME
*    .. External Subroutines ..
      EXTERNAL          XERBLA
*    .. Intrinsic Functions ..
      INTRINSIC          MAX
*    .. Local Scalars ..
      LOGICAL          UPPER
      INTEGER          I, INFO, J, K, NROWA
      DOUBLE PRECISION  TEMP1, TEMP2
*    .. Parameters ..
      DOUBLE PRECISION  ONE          , ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*    ..
*    .. Executable Statements ..
*
*    Set NROWA as the number of rows of A.
*
      IF( LSAME( SIDE, 'L' ) )THEN
        NROWA = M
      ELSE
        NROWA = N
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
*    Test the input parameters.
*
      INFO = 0
      IF(      ( .NOT.LSAME( SIDE, 'L' ) ).AND.
$          ( .NOT.LSAME( SIDE, 'R' ) ) )THEN
        INFO = 1
      ELSE IF( ( .NOT.UPPER ) )AND.
$          ( .NOT.LSAME( UPLO, 'L' ) ) )THEN
        INFO = 2

```



```

      ELSE IF( M .LT.0 ) THEN
        INFO = 3
      ELSE IF( N .LT.0 ) THEN
        INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
        INFO = 7
      ELSE IF( LDB.LT.MAX( 1, M ) ) THEN
        INFO = 9
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
        INFO = 12
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DSYMM ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$      ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
        IF( BETA.EQ.ZERO ) THEN
          DO 20, J = 1, N
            DO 10, I = 1, M
              C( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
        ELSE
          DO 40, J = 1, N
            DO 30, I = 1, M
              C( I, J ) = BETA*C( I, J )
30          CONTINUE
40          CONTINUE
        END IF
        RETURN
      END IF
*
*   Start the operations.
*
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*       Form  C := alpha*A*B + beta*C.
*
        IF( UPPER ) THEN
          DO 70, J = 1, N
            DO 60, I = 1, M

```

```

        TEMP1 = ALPHA*B( I, J )
        TEMP2 = ZERO
        DO 50, K = 1, I - 1
            C( K, J ) = C( K, J ) + TEMP1      *A( K, I )
            TEMP2      = TEMP2      + B( K, J )*A( K, I )
50      CONTINUE
        IF( BETA.EQ.ZERO )THEN
            C( I, J ) = TEMP1*A( I, I ) + ALPHA*TEMP2
        ELSE
            C( I, J ) = BETA *C( I, J ) +
$          TEMP1*A( I, I ) + ALPHA*TEMP2
        END IF
60      CONTINUE
70      CONTINUE
    ELSE
        DO 100, J = 1, N
            DO 90, I = M, 1, -1
                TEMP1 = ALPHA*B( I, J )
                TEMP2 = ZERO
                DO 80, K = I + 1, M
                    C( K, J ) = C( K, J ) + TEMP1      *A( K, I )
                    TEMP2      = TEMP2      + B( K, J )*A( K, I )
80      CONTINUE
                IF( BETA.EQ.ZERO )THEN
                    C( I, J ) = TEMP1*A( I, I ) + ALPHA*TEMP2
                ELSE
                    C( I, J ) = BETA *C( I, J ) +
$          TEMP1*A( I, I ) + ALPHA*TEMP2
                END IF
90      CONTINUE
100     CONTINUE
        END IF
    ELSE
*
*      Form C := alpha*B*A + beta*C.
*
        DO 170, J = 1, N
            TEMP1 = ALPHA*A( J, J )
            IF( BETA.EQ.ZERO )THEN
                DO 110, I = 1, M
                    C( I, J ) = TEMP1*B( I, J )
110     CONTINUE
            ELSE
                DO 120, I = 1, M
                    C( I, J ) = BETA*C( I, J ) + TEMP1*B( I, J )
120     CONTINUE
            END IF
            DO 140, K = 1, J - 1
                IF( UPPER )THEN
                    TEMP1 = ALPHA*A( K, J )

```

```

        ELSE
            TEMP1 = ALPHA*A( J, K )
        END IF
        DO 130, I = 1, M
            C( I, J ) = C( I, J ) + TEMP1*B( I, K )
130      CONTINUE
140      CONTINUE
        DO 160, K = J + 1, N
            IF( UPPER )THEN
                TEMP1 = ALPHA*A( J, K )
            ELSE
                TEMP1 = ALPHA*A( K, J )
            END IF
            DO 150, I = 1, M
                C( I, J ) = C( I, J ) + TEMP1*B( I, K )
150          CONTINUE
160        CONTINUE
170      CONTINUE
    END IF
*
    RETURN
*
*   End of DSYMM .
*
    END

```

— BLAS 3 dsymm —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dsymm (side uplo m n alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
              (type (double-float) beta alpha)
              (type fixnum ldc ldb$ lda n m)
              (type character uplo side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (uplo character uplo-%data% uplo-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%)
       (c double-float c-%data% c-%offset%))
      (prog ((temp1 0.0) (temp2 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
             (upper nil))
        (declare (type (double-float) temp1 temp2)
                  (type fixnum i info j k nrowa)

```

```

        (type (member t nil) upper))
(cond
  ((char-equal side #\L)
   (setf nrowa m))
  (t
   (setf nrowa n)))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
  ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
   (setf info 1))
  ((and (not upper) (not (char-equal uplo #\L)))
   (setf info 2))
  ((< m 0)
   (setf info 3))
  ((< n 0)
   (setf info 4))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 7))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 9))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 12)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DSYMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
    ((= beta zero)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
      (tagbody
       (setf (f2cl-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *))
                           c-%offset%)
              zero))))))
    (t
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody

```



```

a-%offset%))))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:fref b-%data%
        (k j)
        ((1 ldb$) (1 *))
        b-%offset%)
      (f2cl-lib:fref a-%data%
        (k i)
        ((1 lda) (1 *))
        a-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+
        (* temp1
          (f2cl-lib:fref a-%data%
            (i i)
            ((1 lda) (1 *))
            a-%offset%))
        (* alpha temp2))))
  (t
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+
        (* beta
          (f2cl-lib:fref c-%data%
            (i j)
            ((1 ldc) (1 *))
            c-%offset%))
        (* temp1
          (f2cl-lib:fref a-%data%
            (i i)
            ((1 lda) (1 *))
            a-%offset%))
        (* alpha temp2))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
        ((> i 1) nil)
        (tagbody
          (setf temp1
            (f2cl-lib:fref a-%data%
              (i i)
              ((1 lda) (1 *))
              a-%offset%))
            (f2cl-lib:fref b-%data%
              (k j)
              ((1 ldb$) (1 *))
              b-%offset%)
            (* temp1
              (f2cl-lib:fref a-%data%
                (i i)
                ((1 lda) (1 *))
                a-%offset%))
            (* alpha temp2))))))

```

```

(* alpha
  (f2cl-lib:fref b-%data%
                 (i j)
                 ((1 ldb$) (1 *))
                 b-%offset%)))
(setf temp2 zero)
(f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
               (f2cl-lib:int-add k 1))
              (> k m) nil)
(tagbody
 (setf (f2cl-lib:fref c-%data%
                     (k j)
                     ((1 ldc) (1 *))
                     c-%offset%)
       (+
        (f2cl-lib:fref c-%data%
                       (k j)
                       ((1 ldc) (1 *))
                       c-%offset%)
        (* temp1
          (f2cl-lib:fref a-%data%
                        (k i)
                        ((1 lda) (1 *))
                        a-%offset%))))))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:fref b-%data%
                     (k j)
                     ((1 ldb$) (1 *))
                     b-%offset%)
      (f2cl-lib:fref a-%data%
                     (k i)
                     ((1 lda) (1 *))
                     a-%offset%))))))
(cond
 ((= beta zero)
  (setf (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *))
                      c-%offset%)
        (+
         (* temp1
           (f2cl-lib:fref a-%data%
                         (i i)
                         ((1 lda) (1 *))
                         a-%offset%))
         (* alpha temp2))))))
(t
 (setf (f2cl-lib:fref c-%data%

```

```

                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
(+
  (* beta
    (f2cl-lib:fref c-%data%
                   (i j)
                   ((1 ldc) (1 *))
                   c-%offset%))
    (* temp1
      (f2cl-lib:fref a-%data%
                     (i i)
                     ((1 lda) (1 *))
                     a-%offset%))
      (* alpha temp2))))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
                         (j j)
                         ((1 lda) (1 *))
                         a-%offset%)))
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                (* temp1
                  (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%))))))
          (t
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i m) nil)
              (tagbody
                (setf (f2cl-lib:fref c-%data%
                                     (i j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                  (+
                    (* beta
                      (f2cl-lib:fref c-%data%

```



```

                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
      (* temp1
        (f2cl-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *))
          b-%offset%)))))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  ((> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
   nil)
(tagbody
  (cond
    (upper
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
            (k j)
            ((1 lda) (1 *))
            a-%offset%))))))
    (t
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
            (j k)
            ((1 lda) (1 *))
            a-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)
        (* temp1
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%)))))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
  ((> k n) nil)
  (tagbody
    (cond
      (upper
        (setf temp1

```

```

(* alpha
  (f2cl-lib:fref a-%data%
                (j k)
                ((1 lda) (1 *))
                a-%offset%)))
(t
  (setf temp1
    (* alpha
      (f2cl-lib:fref a-%data%
                    (k j)
                    ((1 lda) (1 *))
                    a-%offset%))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
    (+
      (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (* temp1
        (f2cl-lib:fref b-%data%
                      (i k)
                      ((1 ldb$) (1 *))
                      b-%offset%))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil nil)))

```

dsyr2k BLAS

— dsyr2k.input —

```

)set break resume
)sys rm -f dsyr2k.output
)spool dsyr2k.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— dsyr2k.help —

=====

dsyr2k examples

=====

=====

Man Page Details

=====

NAME

DSYR2K - perform one of the symmetric rank 2k operations C
:= alpha*A*B' + alpha*B*A' + beta*C,

SYNOPSIS

SUBROUTINE DSYR2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
 BETA, C, LDC)

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDB, LDC

DOUBLE PRECISION ALPHA, BETA

DOUBLE PRECISION A(LDA, *), B(LDB, *), C(
 LDC, *)

PURPOSE

DSYR2K performs one of the symmetric rank 2k operations

or

C := alpha*A'*B + alpha*B'*A + beta*C,

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part

of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $C := \alpha * A * B' + \alpha * B * A' + \beta * C$.

TRANS = 'T' or 't' $C := \alpha * A' * B + \alpha * B' * A + \beta * C$.

TRANS = 'C' or 'c' $C := \alpha * A' * B + \alpha * B' * A + \beta * C$.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

-
DOUBLE PRECISION array of DIMENSION (LDA, ka), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(

1, n), otherwise LDA must be at least $\max(1, k)$.
 Unchanged on exit.

kb is

B -
 DOUBLE PRECISION array of DIMENSION (LDB, kb), where
 k when TRANS = 'N' or 'n', and is n otherwise.
 Before entry with TRANS = 'N' or 'n', the leading
 n by k part of the array B must contain the matrix
 B, otherwise the leading k by n part of the array
 B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.
 On entry, LDB specifies the first dimension of B as
 declared in the calling (sub) program. When
 TRANS = 'N' or 'n' then LDB must be at least $\max(1, n)$,
 otherwise LDB must be at least $\max(1, k)$.
 Unchanged on exit.

BETA - DOUBLE PRECISION.
 On entry, BETA specifies the scalar beta. Unchanged
 on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n).
 Before entry with UPLO = 'U' or 'u', the leading
 n by n upper triangular part of the array C must con-
 tain the upper triangular part of the symmetric
 matrix and the strictly lower triangular part of C
 is not referenced. On exit, the upper triangular
 part of the array C is overwritten by the upper tri-
 angular part of the updated matrix. Before entry
 with UPLO = 'L' or 'l', the leading n by n lower
 triangular part of the array C must contain the lower
 triangular part of the symmetric matrix and the
 strictly upper triangular part of C is not refer-
 enced. On exit, the lower triangular part of the
 array C is overwritten by the lower triangular part
 of the updated matrix.

LDC - INTEGER.
 On entry, LDC specifies the first dimension of C as
 declared in the calling (sub) program. LDC
 must be at least $\max(1, n)$. Unchanged on exit.

```

      SUBROUTINE DSQR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
$      BETA, C, LDC )
*      .. Scalar Arguments ..
      CHARACTER*1      UPLO, TRANS
      INTEGER          N, K, LDA, LDB, LDC
      DOUBLE PRECISION ALPHA, BETA
*      .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), B( LDB, * ), C( LDC, * )
*
*      ..
*
*      Level 3 Blas routine.
*
*      -- Written on 8-February-1989.
*      Jack Dongarra, Argonne National Laboratory.
*      Iain Duff, AERE Harwell.
*      Jeremy Du Croz, Numerical Algorithms Group Ltd.
*      Sven Hammarling, Numerical Algorithms Group Ltd.
*
*      .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL          LSAME
*      .. External Subroutines ..
      EXTERNAL          XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC          MAX
*      .. Local Scalars ..
      LOGICAL          UPPER
      INTEGER          I, INFO, J, L, NROWA
      DOUBLE PRECISION TEMP1, TEMP2
*      .. Parameters ..
      DOUBLE PRECISION ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      IF( LSAME( TRANS, 'N' ) )THEN
         NROWA = N
      ELSE
         NROWA = K
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
      INFO = 0
      IF( ( .NOT.UPPER ) )AND.
$      ( .NOT.LSAME( UPLO, 'L' ) ) )THEN
         INFO = 1

```

```

      ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ).AND.
$           ( .NOT.LSAME( TRANS, 'T' ) ).AND.
$           ( .NOT.LSAME( TRANS, 'C' ) ) ) THEN
          INFO = 2
      ELSE IF( N .LT.0 ) THEN
          INFO = 3
      ELSE IF( K .LT.0 ) THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
          INFO = 7
      ELSE IF( LDB.LT.MAX( 1, NROWA ) ) THEN
          INFO = 9
      ELSE IF( LDC.LT.MAX( 1, N ) ) THEN
          INFO = 12
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DSYR2K', INFO )
          RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.
$       ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
$       RETURN
*
*   And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
          IF( UPPER ) THEN
              IF( BETA.EQ.ZERO ) THEN
                  DO 20, J = 1, N
                      DO 10, I = 1, J
                          C( I, J ) = ZERO
10                     CONTINUE
20                     CONTINUE
              ELSE
                  DO 40, J = 1, N
                      DO 30, I = 1, J
                          C( I, J ) = BETA*C( I, J )
30                     CONTINUE
40                     CONTINUE
              END IF
          ELSE
              IF( BETA.EQ.ZERO ) THEN
                  DO 60, J = 1, N
                      DO 50, I = J, N
                          C( I, J ) = ZERO
50                     CONTINUE
60                     CONTINUE
              
```

```

ELSE
    DO 80, J = 1, N
        DO 70, I = J, N
            C( I, J ) = BETA*C( I, J )
70        CONTINUE
80        CONTINUE
    END IF
END IF
RETURN
END IF

*
*   Start the operations.
*
IF( LSAME( TRANS, 'N' ) )THEN
*
*   Form C := alpha*A*B' + alpha*B*A' + C.
*
IF( UPPER )THEN
    DO 130, J = 1, N
        IF( BETA.EQ.ZERO )THEN
            DO 90, I = 1, J
                C( I, J ) = ZERO
90            CONTINUE
        ELSE IF( BETA.NE.ONE )THEN
            DO 100, I = 1, J
                C( I, J ) = BETA*C( I, J )
100           CONTINUE
        END IF
        DO 120, L = 1, K
            IF( ( A( J, L ).NE.ZERO ).OR.
$              ( B( J, L ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*B( J, L )
                TEMP2 = ALPHA*A( J, L )
                DO 110, I = 1, J
                    C( I, J ) = C( I, J ) +
$                      A( I, L )*TEMP1 + B( I, L )*TEMP2
110           CONTINUE
                END IF
120           CONTINUE
130           CONTINUE
    ELSE
        DO 180, J = 1, N
            IF( BETA.EQ.ZERO )THEN
                DO 140, I = J, N
                    C( I, J ) = ZERO
140           CONTINUE
            ELSE IF( BETA.NE.ONE )THEN
                DO 150, I = J, N
                    C( I, J ) = BETA*C( I, J )
150           CONTINUE

```



```

        END IF
        DO 170, L = 1, K
            IF( ( A( J, L ).NE.ZERO ).OR.
$           ( B( J, L ).NE.ZERO ) )THEN
                TEMP1 = ALPHA*B( J, L )
                TEMP2 = ALPHA*A( J, L )
                DO 160, I = J, N
                    C( I, J ) = C( I, J ) +
$                    A( I, L )*TEMP1 + B( I, L )*TEMP2
160                CONTINUE
            END IF
170        CONTINUE
180    CONTINUE
    END IF
    ELSE
*
*      Form C := alpha*A'*B + alpha*B'*A + C.
*
        IF( UPPER )THEN
            DO 210, J = 1, N
                DO 200, I = 1, J
                    TEMP1 = ZERO
                    TEMP2 = ZERO
                    DO 190, L = 1, K
                        TEMP1 = TEMP1 + A( L, I )*B( L, J )
                        TEMP2 = TEMP2 + B( L, I )*A( L, J )
190                    CONTINUE
                    IF( BETA.EQ.ZERO )THEN
                        C( I, J ) = ALPHA*TEMP1 + ALPHA*TEMP2
                    ELSE
$                        C( I, J ) = BETA *C( I, J ) +
                        ALPHA*TEMP1 + ALPHA*TEMP2
200                    CONTINUE
210                CONTINUE
            ELSE
                DO 240, J = 1, N
                    DO 230, I = J, N
                        TEMP1 = ZERO
                        TEMP2 = ZERO
                        DO 220, L = 1, K
                            TEMP1 = TEMP1 + A( L, I )*B( L, J )
                            TEMP2 = TEMP2 + B( L, I )*A( L, J )
220                        CONTINUE
                        IF( BETA.EQ.ZERO )THEN
                            C( I, J ) = ALPHA*TEMP1 + ALPHA*TEMP2
                        ELSE
$                            C( I, J ) = BETA *C( I, J ) +
                            ALPHA*TEMP1 + ALPHA*TEMP2
                        END IF

```

```

230          CONTINUE
240          CONTINUE
      END IF
END IF
*
      RETURN
*
*   End of DSYR2K.
*
      END

```

— BLAS 3 dsyr2k —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dsyr2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array double-float (*)) c b a)
              (type (double-float) beta alpha)
              (type fixnum ldc ldb$ lda k n)
              (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%)
       (c double-float c-%data% c-%offset%))
      (prog ((temp1 0.0) (temp2 0.0) (i 0) (info 0) (j 0) (l 0) (nrowa 0)
             (upper nil))
        (declare (type (double-float) temp1 temp2)
                  (type fixnum i info j l nrowa)
                  (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N))
                (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info 2))

```

```

((< n 0)
 (setf info 3))
((< k 0)
 (setf info 4))
((< lda (max (the fixnum 1) (the fixnum nrowa)))
 (setf info 7))
((< ldb$
 (max (the fixnum 1) (the fixnum nrowa)))
 (setf info 9))
((< ldc (max (the fixnum 1) (the fixnum n)))
 (setf info 12)))
(cond
 ((/= info 0)
 (error
  " ** On entry to ~a parameter number ~a had an illegal value~%"
  "DSYR2K" info)
 (go end_label)))
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
 (go end_label))
(cond
 ((= alpha zero)
 (cond
 (upper
 (cond
 ((= beta zero)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j n) nil)
 (tagbody
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i j) nil)
 (tagbody
 (setf (f2cl-lib:fref c-%data%
 (i j)
 ((1 ldc) (1 *))
 c-%offset%)
 zero))))))
 (t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j n) nil)
 (tagbody
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i j) nil)
 (tagbody
 (setf (f2cl-lib:fref c-%data%
 (i j)
 ((1 ldc) (1 *))
 c-%offset%)
 (* beta
 (f2cl-lib:fref c-%data%
 (i j)

```

```

((1 ldc) (1 *))
c-%offset%)))))))))

(t
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
          (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)
              zero))))))

(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
      (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%)
          (* beta
            (f2cl-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *))
                           c-%offset%))))))

  (go end_label)))
(cond
  ((char-equal trans #\N)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (cond
            ((= beta zero)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i j) nil)
              (tagbody
                (setf (f2cl-lib:fref c-%data%
                                     (i j)
                                     ((1 ldc) (1 *))
                                     c-%offset%)
                  zero))))))

```

```

( (/ = beta one)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i j) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (* beta
          (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
  ((> l k) nil)
  (tagbody
    (cond
      ((or (/ = (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
        (/ = (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
      (setf temp1
        (* alpha
          (f2cl-lib:fref b-%data%
                        (j 1)
                        ((1 ldb$) (1 *))
                        b-%offset%)))
      (setf temp2
        (* alpha
          (f2cl-lib:fref a-%data%
                        (j 1)
                        ((1 lda) (1 *))
                        a-%offset%)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i j) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
            (+
              (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)
              (*
                (f2cl-lib:fref a-%data%
                              (i 1)
                              ((1 lda) (1 *))
                              a-%offset%)
                temp1)
              (*

```

```

                                (f2cl-lib:fref b-%data%
                                (i 1)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                                temp2))))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j n) nil)
(tagbody
(cond
  (= beta zero)
  (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
              zero))))
  (/= beta one)
  (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
              (* beta
               (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              (> l k) nil)
(tagbody
(cond
  ((or (/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
        (/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
   (setf temp1
          (* alpha
             (f2cl-lib:fref b-%data%
                             (j 1)
                             ((1 ldb$) (1 *))
                             b-%offset%)))
   (setf temp2
          (* alpha
             (f2cl-lib:fref a-%data%
                             (j 1)
                             ((1 lda) (1 *))
                             a-%offset%)))

```

```

(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
    (+
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)
      (*
        (f2cl-lib:fref a-%data%
          (i 1)
          ((1 lda) (1 *))
          a-%offset%)
        temp1)
      (*
        (f2cl-lib:fref b-%data%
          (i 1)
          ((1 ldb$) (1 *))
          b-%offset%)
        temp2)))))))))
(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i j) nil)
          (tagbody
            (setf temp1 zero)
            (setf temp2 zero)
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              (> l k) nil)
              (tagbody
                (setf temp1
                  (+ temp1
                    (*
                      (f2cl-lib:fref a-%data%
                        (l i)
                        ((1 lda) (1 *))
                        a-%offset%)
                      (f2cl-lib:fref b-%data%
                        (l j)
                        ((1 ldb$) (1 *))
                        b-%offset%))))
                  (setf temp2

```

```

(+ temp2
  (*
    (f2cl-lib:fref b-%data%
      (1 i)
      ((1 ldb$) (1 *)))
    b-%offset%)
    (f2cl-lib:fref a-%data%
      (1 j)
      ((1 lda) (1 *)))
    a-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *)))
      c-%offset%)
    (+ (* alpha temp1) (* alpha temp2))))
(t
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *)))
    c-%offset%)
    (+
      (* beta
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *)))
          c-%offset%)
        (* alpha temp1)
        (* alpha temp2)))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf temp1 zero)
          (setf temp2 zero)
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
            ((> l k) nil)
            (tagbody
              (setf temp1
                (+ temp1
                  (*
                    (f2cl-lib:fref a-%data%
                      (1 i)
                      ((1 lda) (1 *)))
                      a-%offset%)
                    (f2cl-lib:fref b-%data%

```



```

                                (1 j)
                                ((1 ldb$) (1 *))
                                b-%offset%)))
      (setf temp2
        (+ temp2
          (*
            (f2cl-lib:fref b-%data%
                          (1 i)
                          ((1 ldb$) (1 *))
                          b-%offset%)
            (f2cl-lib:fref a-%data%
                          (1 j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
    (cond
      ((= beta zero)
        (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
              (+ (* alpha temp1) (* alpha temp2))))
      (t
        (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
              (+
                (* beta
                  (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                (* alpha temp1)
                (* alpha temp2))))))
    end_label
    (return (values nil nil nil nil nil nil nil nil nil nil))))

```

dsyrk BLAS

— dsyrk.input —

```

)set break resume
)sys rm -f dsyrk.output
)spool dsyrk.output

```

```
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dsyrk.help —

```
=====
dsyrk examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DSYRK - perform one of the symmetric rank k operations C
 $C := \alpha A A' + \beta C,$

SYNOPSIS

SUBROUTINE DSYRK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
 C, LDC)

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDC

DOUBLE PRECISION ALPHA, BETA

DOUBLE PRECISION A(LDA, *), C(LDC, *)

PURPOSE

DSYRK performs one of the symmetric rank k operations

or

$C := \alpha A' A + \beta C,$

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or

lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $C := \alpha * A * A' + \beta * C$.

TRANS = 'T' or 't' $C := \alpha * A' * A + \beta * C$.

TRANS = 'C' or 'c' $C := \alpha * A' * A + \beta * C$.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A -

DOUBLE PRECISION array of DIMENSION (LDA, ka), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When

TRANS = 'N' or 'n' then LDA must be at least $\max(1, n)$, otherwise LDA must be at least $\max(1, k)$. Unchanged on exit.

BETA - DOUBLE PRECISION.
On entry, BETA specifies the scalar beta. Unchanged on exit.

C - DOUBLE PRECISION array of DIMENSION (LDC, n).
Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, n)$. Unchanged on exit.

— dsyrk.f —

```

SUBROUTINE DSYRK ( UPLO, TRANS, N, K, ALPHA, A, LDA,
$                BETA, C, LDC )
*   .. Scalar Arguments ..
CHARACTER*1      UPLO, TRANS
INTEGER          N, K, LDA, LDC
DOUBLE PRECISION ALPHA, BETA
*   .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), C( LDC, * )
*
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.

```

```

*      Jeremy Du Croz, Numerical Algorithms Group Ltd.
*      Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*      .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
*      .. External Subroutines ..
EXTERNAL         XERBLA
*      .. Intrinsic Functions ..
INTRINSIC        MAX
*      .. Local Scalars ..
LOGICAL          UPPER
INTEGER          I, INFO, J, L, NROWA
DOUBLE PRECISION TEMP
*      .. Parameters ..
DOUBLE PRECISION ONE ,          ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      IF( LSAME( TRANS, 'N' ) )THEN
        NROWA = N
      ELSE
        NROWA = K
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
      INFO = 0
      IF(          ( .NOT.UPPER                      ).AND.
$          ( .NOT.LSAME( UPLO , 'L' ) )          )THEN
        INFO = 1
      ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ).AND.
$          ( .NOT.LSAME( TRANS, 'T' ) ).AND.
$          ( .NOT.LSAME( TRANS, 'C' ) )          )THEN
        INFO = 2
      ELSE IF( N .LT.0                      )THEN
        INFO = 3
      ELSE IF( K .LT.0                      )THEN
        INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
        INFO = 7
      ELSE IF( LDC.LT.MAX( 1, N          ) )THEN
        INFO = 10
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'DSYRK ', INFO )
        RETURN

```

```

        END IF
*
*   Quick return if possible.
*
        IF( ( N.EQ.0 ).OR.
$      ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   And when  alpha.eq.zero.
*
        IF( ALPHA.EQ.ZERO )THEN
            IF( UPPER )THEN
                IF( BETA.EQ.ZERO )THEN
                    DO 20, J = 1, N
                        DO 10, I = 1, J
                            C( I, J ) = ZERO
10                        CONTINUE
20                        CONTINUE
                ELSE
                    DO 40, J = 1, N
                        DO 30, I = 1, J
                            C( I, J ) = BETA*C( I, J )
30                        CONTINUE
40                        CONTINUE
                END IF
            ELSE
                IF( BETA.EQ.ZERO )THEN
                    DO 60, J = 1, N
                        DO 50, I = J, N
                            C( I, J ) = ZERO
50                        CONTINUE
60                        CONTINUE
                ELSE
                    DO 80, J = 1, N
                        DO 70, I = J, N
                            C( I, J ) = BETA*C( I, J )
70                        CONTINUE
80                        CONTINUE
                END IF
            END IF
            RETURN
        END IF
*
*   Start the operations.
*
        IF( LSAME( TRANS, 'N' ) )THEN
*
*       Form  C := alpha*A*A' + beta*C.
*
            IF( UPPER )THEN

```

```

DO 130, J = 1, N
  IF( BETA.EQ.ZERO )THEN
    DO 90, I = 1, J
      C( I, J ) = ZERO
90    CONTINUE
  ELSE IF( BETA.NE.ONE )THEN
    DO 100, I = 1, J
      C( I, J ) = BETA*C( I, J )
100    CONTINUE
  END IF
  DO 120, L = 1, K
    IF( A( J, L ).NE.ZERO )THEN
      TEMP = ALPHA*A( J, L )
      DO 110, I = 1, J
        C( I, J ) = C( I, J ) + TEMP*A( I, L )
110      CONTINUE
      END IF
120    CONTINUE
130  CONTINUE
ELSE
  DO 180, J = 1, N
    IF( BETA.EQ.ZERO )THEN
      DO 140, I = J, N
        C( I, J ) = ZERO
140      CONTINUE
    ELSE IF( BETA.NE.ONE )THEN
      DO 150, I = J, N
        C( I, J ) = BETA*C( I, J )
150      CONTINUE
    END IF
    DO 170, L = 1, K
      IF( A( J, L ).NE.ZERO )THEN
        TEMP = ALPHA*A( J, L )
        DO 160, I = J, N
          C( I, J ) = C( I, J ) + TEMP*A( I, L )
160        CONTINUE
        END IF
      END IF
170    CONTINUE
180  CONTINUE
  END IF
ELSE
*
*   Form C := alpha*A'*A + beta*C.
*
  IF( UPPER )THEN
    DO 210, J = 1, N
      DO 200, I = 1, J
        TEMP = ZERO
        DO 190, L = 1, K
          TEMP = TEMP + A( L, I )*A( L, J )

```

```

190          CONTINUE
            IF( BETA.EQ.ZERO )THEN
              C( I, J ) = ALPHA*TEMP
            ELSE
              C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
            END IF
200          CONTINUE
210          CONTINUE
        ELSE
          DO 240, J = 1, N
            DO 230, I = J, N
              TEMP = ZERO
              DO 220, L = 1, K
                TEMP = TEMP + A( L, I )*A( L, J )
220              CONTINUE
              IF( BETA.EQ.ZERO )THEN
                C( I, J ) = ALPHA*TEMP
              ELSE
                C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
              END IF
230            CONTINUE
240          CONTINUE
        END IF
      END IF
*
      RETURN
*
*   End of DSYRK .
*
      END

```

— BLAS 3 dsyrk —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dsyrk (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array double-float (*)) c a)
              (type (double-float) beta alpha)
              (type fixnum ldc lda k n)
              (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (c double-float c-%data% c-%offset%))

```



```

(prog ((temp 0.0) (i 0) (info 0) (j 0) (l 0) (nrowa 0) (upper nil))
  (declare (type (double-float) temp)
            (type fixnum i info j l nrowa)
            (type (member t nil) upper))
  (cond
    ((char-equal trans #\N)
     (setf nrowa n))
    (t
     (setf nrowa k)))
  (setf upper (char-equal uplo #\U))
  (setf info 0)
  (cond
    ((and (not upper) (not (char-equal uplo #\L)))
     (setf info 1))
    ((and (not (char-equal trans #\N))
          (not (char-equal trans #\T))
          (not (char-equal trans #\C)))
     (setf info 2))
    ((< n 0)
     (setf info 3))
    ((< k 0)
     (setf info 4))
    ((< lda (max (the fixnum 1) (the fixnum nrowa)))
     (setf info 7))
    ((< ldc (max (the fixnum 1) (the fixnum n)))
     (setf info 10)))
  (cond
    ((/= info 0)
     (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DSYRK" info)
     (go end_label)))
  (if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
      (go end_label))
  (cond
    ((= alpha zero)
     (cond
      (upper
       (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                       ((> j n) nil)
         (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                        ((> i j) nil)
          (tagbody
           (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%))

```

```

                                zero))))))
(t
  (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
      (> i j) nil)
      (tagbody
        (setf (f2c1-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *)))
              c-%offset%)
          (* beta
            (f2c1-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *)))
            c-%offset%)))))))))
(t
  (cond
    ((= beta zero)
     (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
       (> j n) nil)
     (tagbody
       (f2c1-lib:fdo (i j (f2c1-lib:int-add i 1))
         (> i n) nil)
         (tagbody
           (setf (f2c1-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *)))
                c-%offset%)
              zero))))))
(t
  (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2c1-lib:fdo (i j (f2c1-lib:int-add i 1))
      (> i n) nil)
      (tagbody
        (setf (f2c1-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *)))
              c-%offset%)
          (* beta
            (f2c1-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *)))
            c-%offset%)))))))))
  (go end_label)))
(cond
  ((char-equal trans #\N)

```

```

(cond
  (upper
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
    (tagbody
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i j) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)
              zero))))
        ((/= beta one)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i j) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)
              (* beta
                (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)))))))
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
        ((> l k) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
            (setf temp
              (* alpha
                (f2cl-lib:fref a-%data%
                               (j 1)
                               ((1 lda) (1 *))
                               a-%offset%)))
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i j) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                (+
                  (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))

```

```

                                c-%offset%)
                                (* temp
                                (f2cl-lib:fref a-%data%
                                (i 1)
                                ((1 lda) (1 *))
                                a-%offset%))))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
(tagbody
 (cond
  ((= beta zero)
   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                 ((> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          zero))))
  ((/= beta one)
   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                 ((> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* beta
            (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
              ((> l k) nil)
(tagbody
 (cond
  ((/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
   (setf temp
            (* alpha
              (f2cl-lib:fref a-%data%
                              (j 1)
                              ((1 lda) (1 *))
                              a-%offset%)))
   (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                 ((> i n) nil)
   (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))

```

```

                                c-%offset%)
      (+
        (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *)))
        c-%offset%)

      (* temp
        (f2cl-lib:fref a-%data%
                      (i 1)
                      ((1 lda) (1 *)))
        a-%offset%)))))))))))))

(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i j) nil)
          (tagbody
            (setf temp zero)
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                          ((> l k) nil)
              (tagbody
                (setf temp
                  (+ temp
                    (*
                     (f2cl-lib:fref a-%data%
                                     (l i)
                                     ((1 lda) (1 *)))
                     a-%offset%)
                    (f2cl-lib:fref a-%data%
                                     (l j)
                                     ((1 lda) (1 *)))
                     a-%offset%))))))
              (cond
                ((= beta zero)
                  (setf (f2cl-lib:fref c-%data%
                                       (i j)
                                       ((1 ldc) (1 *)))
                      c-%offset%)
                  (* alpha temp)))
                (t
                  (setf (f2cl-lib:fref c-%data%
                                       (i j)
                                       ((1 ldc) (1 *)))
                      c-%offset%)
                  (+ (* alpha temp)
                     (* beta
                      (f2cl-lib:fref c-%data%

```

```

(i j)
((1 ldc) (1 *))
c-%offset%))))))))))

(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j n) nil)
(tagbody
(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
  ((> i n) nil)
(tagbody
(setf temp zero)
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
  ((> l k) nil)
(tagbody
(setf temp
  (+ temp
    (*
      (f2cl-lib:fref a-%data%
        (l i)
        ((1 lda) (1 *))
        a-%offset%)
      (f2cl-lib:fref a-%data%
        (l j)
        ((1 lda) (1 *))
        a-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (* alpha temp)))
(t
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
    (+ (* alpha temp)
      (* beta
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil))))

```

dtrmm BLAS**— dtrmm.input —**

```

)set break resume
)sys rm -f dtrmm.output
)spool dtrmm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrmm.help —

```

=====
dtrmm examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DTRMM - perform one of the matrix-matrix operations $B := \alpha * \text{op}(A) * B$, or $B := \alpha * B * \text{op}(A)$,

SYNOPSIS

```

SUBROUTINE DTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                  LDA, B, LDB )

```

CHARACTER*1 SIDE, UPLO, TRANSA, DIAG

INTEGER M, N, LDA, LDB

DOUBLE PRECISION ALPHA

DOUBLE PRECISION A(LDA, *), B(LDB, *)

PURPOSE

DTRMM performs one of the matrix-matrix operations

where α is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A)$ is one of

$\text{op}(A) = A$ or $\text{op}(A) = A'$.

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether $\text{op}(A)$ multiplies B from the left or right as follows:

SIDE = 'L' or 'l' B := $\alpha * \text{op}(A) * B$.

SIDE = 'R' or 'r' B := $\alpha * B * \text{op}(A)$.

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' $\text{op}(A) = A$.

TRANSA = 'T' or 't' $\text{op}(A) = A'$.

TRANSA = 'C' or 'c' $\text{op}(A) = A'$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of B. M must

be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

is m

A -

DOUBLE PRECISION array of DIMENSION (LDA, k), where k when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least $\max(1, m)$, when SIDE = 'R' or 'r' then LDA must be at least $\max(1, n)$. Unchanged on exit.

B - DOUBLE PRECISION array of DIMENSION (LDB, n).

Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

LDB - INTEGER.

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least $\max(1, m)$. Unchanged on exit.

```

      SUBROUTINE DTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
$          B, LDB )
*
*   .. Scalar Arguments ..
      CHARACTER*1      SIDE, UPLO, TRANSA, DIAG
      INTEGER          M, N, LDA, LDB
      DOUBLE PRECISION ALPHA
*
*   .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), B( LDB, * )
*
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*   .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*
*   .. External Subroutines ..
      EXTERNAL         XERBLA
*
*   .. Intrinsic Functions ..
      INTRINSIC        MAX
*
*   .. Local Scalars ..
      LOGICAL          LSIDE, NOUNIT, UPPER
      INTEGER          I, INFO, J, K, NROWA
      DOUBLE PRECISION TEMP
*
*   .. Parameters ..
      DOUBLE PRECISION ONE, ZERO
      PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*   ..
*
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      LSIDE = LSAME( SIDE , 'L' )
      IF( LSIDE )THEN
         NROWA = M
      ELSE
         NROWA = N
      END IF
      NOUNIT = LSAME( DIAG , 'N' )
      UPPER = LSAME( UPLO , 'U' )
*
      INFO = 0
      IF( ( .NOT.LSIDE ) .AND.
$        ( .NOT.LSAME( SIDE , 'R' ) ) )THEN

```

```

        INFO = 1
        ELSE IF( ( .NOT.UPPER                      ).AND.
$             ( .NOT.LSAME( UPLO , 'L' ) ) ) THEN
            INFO = 2
        ELSE IF( ( .NOT.LSAME( TRANSA, 'N' ) ).AND.
$             ( .NOT.LSAME( TRANSA, 'T' ) ).AND.
$             ( .NOT.LSAME( TRANSA, 'C' ) ) ) THEN
            INFO = 3
        ELSE IF( ( .NOT.LSAME( DIAG , 'U' ) ).AND.
$             ( .NOT.LSAME( DIAG , 'N' ) ) ) THEN
            INFO = 4
        ELSE IF( M .LT.0 ) THEN
            INFO = 5
        ELSE IF( N .LT.0 ) THEN
            INFO = 6
        ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
            INFO = 9
        ELSE IF( LDB.LT.MAX( 1, M ) ) THEN
            INFO = 11
        END IF
        IF( INFO.NE.0 ) THEN
            CALL XERBLA( 'DTRMM ', INFO )
            RETURN
        END IF

*
*   Quick return if possible.
*
        IF( N.EQ.0 )
$           RETURN

*
*   And when  alpha.eq.zero.
*
        IF( ALPHA.EQ.ZERO ) THEN
            DO 20, J = 1, N
                DO 10, I = 1, M
                    B( I, J ) = ZERO
10                CONTINUE
20            CONTINUE
            RETURN
        END IF

*
*   Start the operations.
*
        IF( LSIDE ) THEN
            IF( LSAME( TRANSA, 'N' ) ) THEN

*
*           Form  B := alpha*A*B.
*

                IF( UPPER ) THEN
                    DO 50, J = 1, N

```

```

DO 40, K = 1, M
  IF( B( K, J ).NE.ZERO )THEN
    TEMP = ALPHA*B( K, J )
    DO 30, I = 1, K - 1
      B( I, J ) = B( I, J ) + TEMP*A( I, K )
30    CONTINUE
    IF( NOUNIT )
      $      TEMP = TEMP*A( K, K )
      B( K, J ) = TEMP
    END IF
40    CONTINUE
50    CONTINUE
  ELSE
    DO 80, J = 1, N
      DO 70 K = M, 1, -1
        IF( B( K, J ).NE.ZERO )THEN
          TEMP = ALPHA*B( K, J )
          B( K, J ) = TEMP
          IF( NOUNIT )
            $      B( K, J ) = B( K, J )*A( K, K )
          DO 60, I = K + 1, M
            B( I, J ) = B( I, J ) + TEMP*A( I, K )
60          CONTINUE
          END IF
70          CONTINUE
80          CONTINUE
        END IF
      ELSE
        *
        *      Form B := alpha*A'*B.
        *
        IF( UPPER )THEN
          DO 110, J = 1, N
            DO 100, I = M, 1, -1
              TEMP = B( I, J )
              IF( NOUNIT )
                $      TEMP = TEMP*A( I, I )
              DO 90, K = 1, I - 1
                TEMP = TEMP + A( K, I )*B( K, J )
90              CONTINUE
              B( I, J ) = ALPHA*TEMP
100             CONTINUE
110            CONTINUE
          ELSE
            DO 140, J = 1, N
              DO 130, I = 1, M
                TEMP = B( I, J )
                IF( NOUNIT )
                  $      TEMP = TEMP*A( I, I )
                DO 120, K = I + 1, M

```

```

      TEMP = TEMP + A( K, I ) * B( K, J )
120      CONTINUE
      B( I, J ) = ALPHA * TEMP
130      CONTINUE
140      CONTINUE
      END IF
      END IF
      ELSE
      IF( LSAME( TRANSA, 'N' ) ) THEN
*
*      Form B := alpha*B*A.
*
      IF( UPPER ) THEN
        DO 180, J = N, 1, -1
          TEMP = ALPHA
          IF( NOUNIT )
$            TEMP = TEMP * A( J, J )
          DO 150, I = 1, M
            B( I, J ) = TEMP * B( I, J )
150          CONTINUE
          DO 170, K = 1, J - 1
            IF( A( K, J ).NE.ZERO ) THEN
              TEMP = ALPHA * A( K, J )
              DO 160, I = 1, M
                B( I, J ) = B( I, J ) + TEMP * B( I, K )
160              CONTINUE
            END IF
          CONTINUE
170          CONTINUE
180          CONTINUE
        ELSE
          DO 220, J = 1, N
            TEMP = ALPHA
            IF( NOUNIT )
$              TEMP = TEMP * A( J, J )
            DO 190, I = 1, M
              B( I, J ) = TEMP * B( I, J )
190            CONTINUE
            DO 210, K = J + 1, N
              IF( A( K, J ).NE.ZERO ) THEN
                TEMP = ALPHA * A( K, J )
                DO 200, I = 1, M
                  B( I, J ) = B( I, J ) + TEMP * B( I, K )
200                CONTINUE
              END IF
            CONTINUE
210          CONTINUE
220          CONTINUE
        END IF
      ELSE
*
*      Form B := alpha*B*A'.

```

```

*
      IF( UPPER )THEN
        DO 260, K = 1, N
          DO 240, J = 1, K - 1
            IF( A( J, K ).NE.ZERO )THEN
              TEMP = ALPHA*A( J, K )
              DO 230, I = 1, M
                B( I, J ) = B( I, J ) + TEMP*B( I, K )
230              CONTINUE
            END IF
          CONTINUE
240          TEMP = ALPHA
          IF( NOUNIT )
            $      TEMP = TEMP*A( K, K )
            IF( TEMP.NE.ONE )THEN
              DO 250, I = 1, M
                B( I, K ) = TEMP*B( I, K )
250              CONTINUE
            END IF
          CONTINUE
260        ELSE
          DO 300, K = N, 1, -1
            DO 280, J = K + 1, N
              IF( A( J, K ).NE.ZERO )THEN
                TEMP = ALPHA*A( J, K )
                DO 270, I = 1, M
                  B( I, J ) = B( I, J ) + TEMP*B( I, K )
270                CONTINUE
              END IF
            CONTINUE
280            TEMP = ALPHA
            IF( NOUNIT )
              $      TEMP = TEMP*A( K, K )
              IF( TEMP.NE.ONE )THEN
                DO 290, I = 1, M
                  B( I, K ) = TEMP*B( I, K )
290                CONTINUE
              END IF
            CONTINUE
300          END IF
        END IF
      END IF
*
      RETURN
*
*      End of DTRMM .
*
      END

```

— BLAS 3 dtrmm —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dtrmm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array double-float (*)) b a)
              (type (double-float) alpha)
              (type fixnum ldb$ lda n m)
              (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (uplo character uplo-%data% uplo-%offset%)
       (transa character transa-%data% transa-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0) (lside nil)
              (nounit nil) (upper nil))
        (declare (type (double-float) temp)
                  (type fixnum i info j k nrowa)
                  (type (member t nil) lside nounit upper))
        (setf lside (char-equal side #\L))
        (cond
          (lside
            (setf nrowa m))
          (t
            (setf nrowa n)))
        (setf nounit (char-equal diag #\N))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not lside) (not (char-equal side #\R)))
            (setf info 1))
          ((and (not upper) (not (char-equal uplo #\L)))
            (setf info 2))
          ((and (not (char-equal transa #\N))
                (not (char-equal transa #\T))
                (not (char-equal transa #\C)))
            (setf info 3))
          ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
            (setf info 4))
          ((< m 0)
            (setf info 5))
          ((< n 0)
            (setf info 6))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
            (setf info 7))
          (t
            (setf info 0)))
        (if (eql info 0)
            (return)
            (let ((info (1+ info)))
              (return))))))

```

```
(setf info 9))
(((< ldb$ (max (the fixnum 1) (the fixnum m)))
 (setf info 11)))
(cond
 ((/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "DTRMM" info)
  (go end_label)))
(if (= n 0) (go end_label))
(cond
 ((= alpha zero)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     (> i m) nil)
   (tagbody
    (setf (f2cl-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *)))
      b-%offset%)
      zero))))))
(go end_label)))
(cond
 (lside
  (cond
   ((char-equal transa #\N)
    (cond
     (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
       (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
         (> k m) nil)
       (tagbody
        (cond
         ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
          (setf temp
               (* alpha
                  (f2cl-lib:fref b-%data%
                                  (k j)
                                  ((1 ldb$) (1 *)))
                  b-%offset%)))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i
              (f2cl-lib:int-add k
                (f2cl-lib:int-sub
                 1)))
              nil)
```



```

      (tagbody
        (setf (f2cl-lib:fref b-%data%
                             (i j)
                             ((1 ldb$) (1 *)))
              b-%offset%)
          (+
            (f2cl-lib:fref b-%data%
                           (i j)
                           ((1 ldb$) (1 *)))
              b-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                             (i k)
                             ((1 lda) (1 *)))
              a-%offset%))))))
    (if nount
      (setf temp
        (* temp
          (f2cl-lib:fref a-%data%
                         (k k)
                         ((1 lda) (1 *)))
          a-%offset%)))
      (setf (f2cl-lib:fref b-%data%
                           (k j)
                           ((1 ldb$) (1 *)))
            b-%offset%)
        temp))))))
  (t
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (f2cl-lib:fdo (k m
                    (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
        (> k 1) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
            (setf temp
              (* alpha
                (f2cl-lib:fref b-%data%
                               (k j)
                               ((1 ldb$) (1 *)))
                b-%offset%)))
            (setf (f2cl-lib:fref b-%data%
                               (k j)
                               ((1 ldb$) (1 *)))
                  b-%offset%)
              temp)
          (if nount
            (setf (f2cl-lib:fref b-%data%

```

```

(k j)
((1 ldb$) (1 *))
b-%offset%)

(*
(f2cl-lib:fref b-%data%
(k j)
((1 ldb$) (1 *))
b-%offset%)
(f2cl-lib:fref a-%data%
(k k)
((1 lda) (1 *))
a-%offset%)))
(f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
(f2cl-lib:int-add i 1))
(> i m) nil)
(tagbody
(setf (f2cl-lib:fref b-%data%
(i j)
((1 ldb$) (1 *))
b-%offset%)
(
(f2cl-lib:fref b-%data%
(i j)
((1 ldb$) (1 *))
b-%offset%)
(* temp
(f2cl-lib:fref a-%data%
(i k)
((1 lda) (1 *))
a-%offset%)))))))))))))
(t
(cond
(upper
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(f2cl-lib:fdo (i m
(f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
(> i 1) nil)
(tagbody
(setf temp
(f2cl-lib:fref b-%data%
(i j)
((1 ldb$) (1 *))
b-%offset%)
(if nunit
(setf temp
(* temp
(f2cl-lib:fref a-%data%
(i i)

```

```

((1 lda) (1 *))
a-%offset%))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add i
      (f2cl-lib:int-sub
        1)))
    nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2cl-lib:fref a-%data%
          (k i)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:fref b-%data%
          (k j)
          ((1 ldb$) (1 *))
          b-%offset%))))))
(setf (f2cl-lib:fref b-%data%
  (i j)
  ((1 ldb$) (1 *))
  b-%offset%)
  (* alpha temp))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%))
      (if nounit
        (setf temp
          (* temp
            (f2cl-lib:fref a-%data%
              (i i)
              ((1 lda) (1 *))
              a-%offset%))))
        (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
          (f2cl-lib:int-add k 1))
            (> k m) nil)
          (tagbody
            (setf temp
              (+ temp
```

```

(*
  (f2c1-lib:fref a-%data%
    (k i)
    ((1 lda) (1 *))
    a-%offset%)
  (f2c1-lib:fref b-%data%
    (k j)
    ((1 ldb$) (1 *))
    b-%offset%))))))
(setf (f2c1-lib:fref b-%data%
  (i j)
  ((1 ldb$) (1 *))
  b-%offset%)
  (* alpha temp)))))))))
(t
  (cond
    ((char-equal transa #\N)
      (cond
        (upper
          (f2c1-lib:fdo (j n (f2c1-lib:int-add j (f2c1-lib:int-sub 1)))
            ((> j 1) nil)
            (tagbody
              (setf temp alpha)
              (if nounit
                (setf temp
                  (* temp
                    (f2c1-lib:fref a-%data%
                      (j j)
                      ((1 lda) (1 *))
                      a-%offset%))))
                (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
                  ((> i m) nil)
                  (tagbody
                    (setf (f2c1-lib:fref b-%data%
                      (i j)
                      ((1 ldb$) (1 *))
                      b-%offset%)
                      (* temp
                        (f2c1-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *))
                          b-%offset%))))
                    (f2c1-lib:fdo (k 1 (f2c1-lib:int-add k 1))
                      ((> k
                        (f2c1-lib:int-add j
                          (f2c1-lib:int-sub 1)))
                        nil)
                      (tagbody
                        (cond
                          ((/= (f2c1-lib:fref a (k j) ((1 lda) (1 *))) zero)

```

```

(setf temp
  (* alpha
    (f2cl-lib:fref a-%data%
      (k j)
      ((1 lda) (1 *))
      a-%offset%)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref b-%data%
      (i j)
      ((1 ldb$) (1 *))
      b-%offset%)
      (+
        (f2cl-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *))
          b-%offset%)
        (* temp
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%)))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp alpha)
      (if nounit
        (setf temp
          (* temp
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
              (i j)
              ((1 ldb$) (1 *))
              b-%offset%)
              (* temp
                (f2cl-lib:fref b-%data%
                  (i j)
                  ((1 ldb$) (1 *))
                  b-%offset%))))
            (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
              (f2cl-lib:int-add k 1))
              ((> k n) nil)

```

```
(tagbody
(cond
  ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
    (setf temp
      (* alpha
        (f2cl-lib:fref a-%data%
          (k j)
          ((1 lda) (1 *))
          a-%offset%)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)
              (+
                (f2cl-lib:fref b-%data%
                  (i j)
                  ((1 ldb$) (1 *))
                  b-%offset%)
                (* temp
                  (f2cl-lib:fref b-%data%
                    (i k)
                    ((1 ldb$) (1 *))
                    b-%offset%))))))))))
(t
(cond
  (upper
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
      ((> k n) nil)
      (tagbody
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j
            (f2cl-lib:int-add k
              (f2cl-lib:int-sub 1)))
            nil)
          (tagbody
            (cond
              ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                (setf temp
                  (* alpha
                    (f2cl-lib:fref a-%data%
                      (j k)
                      ((1 lda) (1 *))
                      a-%offset%)))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
                  (tagbody
                    (setf (f2cl-lib:fref b-%data%
```

```

                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
      (+
        (f2cl-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *))
          b-%offset%)
        (* temp
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%)))))))))
    (setf temp alpha)
    (if nounit
      (setf temp
        (* temp
          (f2cl-lib:fref a-%data%
            (k k)
            ((1 lda) (1 *))
            a-%offset%))))))
    (cond
      ((/= temp one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
              (i k)
              ((1 ldb$) (1 *))
              b-%offset%)
              (* temp
                (f2cl-lib:fref b-%data%
                  (i k)
                  ((1 ldb$) (1 *))
                  b-%offset%)))))))))
    (t
      (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
        ((> k 1) nil)
        (tagbody
          (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
            (f2cl-lib:int-add j 1))
            ((> j n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
                  (setf temp
                    (* alpha
                      (f2cl-lib:fref a-%data%
                        (j k)
                        ((1 lda) (1 *))

```

```

                                a-%offset%)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *)))
          b-%offset%)
      (+
        (f2cl-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *)))
          b-%offset%)
      (* temp
        (f2cl-lib:fref b-%data%
                        (i k)
                        ((1 ldb$) (1 *)))
          b-%offset%)))))))))
(setf temp alpha)
(if nount
  (setf temp
    (* temp
      (f2cl-lib:fref a-%data%
                      (k k)
                      ((1 lda) (1 *)))
        a-%offset%)))
  (cond
    ((/= temp one)
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   (> i m) nil)
     (tagbody
      (setf (f2cl-lib:fref b-%data%
                          (i k)
                          ((1 ldb$) (1 *)))
            b-%offset%)
          (* temp
            (f2cl-lib:fref b-%data%
                            (i k)
                            ((1 ldb$) (1 *)))
              b-%offset%)))))))))
end_label
  (return (values nil nil nil nil nil nil nil nil nil nil))))

```

dtrsm BLAS**— dtrsm.input —**

```

)set break resume
)sys rm -f dtrsm.output
)spool dtrsm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrsm.help —

```

=====
dtrsm examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DTRSM - solve one of the matrix equations $\text{op}(A)X = \alpha B$, or $X\text{op}(A) = \alpha B$,

SYNOPSIS

SUBROUTINE DTRSM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
LDA, B, LDB)

CHARACTER*1 SIDE, UPLO, TRANSA, DIAG

INTEGER M, N, LDA, LDB

DOUBLE PRECISION ALPHA

DOUBLE PRECISION A(LDA, *), B(LDB, *)

PURPOSE

DTRSM solves one of the matrix equations

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A)$ is one of

$\text{op}(A) = A$ or $\text{op}(A) = A'$.

The matrix X is overwritten on B .

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether $\text{op}(A)$ appears on the left or right of X as follows:

SIDE = 'L' or 'l' $\text{op}(A)*X = \alpha*B$.

SIDE = 'R' or 'r' $X*\text{op}(A) = \alpha*B$.

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' $\text{op}(A) = A$.

TRANSA = 'T' or 't' $\text{op}(A) = A'$.

TRANSA = 'C' or 'c' $\text{op}(A) = A'$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M - INTEGER.
 On entry, M specifies the number of rows of B. M must
 be at least zero. Unchanged on exit.

N - INTEGER.
 On entry, N specifies the number of columns of B. N
 must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.
 On entry, ALPHA specifies the scalar alpha. When
 alpha is zero then A is not referenced and B need
 not be set before entry. Unchanged on exit.

is m
 A -
 DOUBLE PRECISION array of DIMENSION (LDA, k), where k
 when SIDE = 'L' or 'l' and is n when SIDE = 'R'
 or 'r'. Before entry with UPLO = 'U' or 'u', the
 leading k by k upper triangular part of the array A
 must contain the upper triangular matrix and the
 strictly lower triangular part of A is not refer-
 enced. Before entry with UPLO = 'L' or 'l', the
 leading k by k lower triangular part of the array A
 must contain the lower triangular matrix and the
 strictly upper triangular part of A is not refer-
 enced. Note that when DIAG = 'U' or 'u', the diag-
 onal elements of A are not referenced either, but
 are assumed to be unity. Unchanged on exit.

LDA - INTEGER.
 On entry, LDA specifies the first dimension of A as
 declared in the calling (sub) program. When SIDE =
 'L' or 'l' then LDA must be at least $\max(1, m)$,
 when SIDE = 'R' or 'r' then LDA must be at least
 $\max(1, n)$. Unchanged on exit.

B - DOUBLE PRECISION array of DIMENSION (LDB, n).
 Before entry, the leading m by n part of the array
 B must contain the right-hand side matrix B,
 and on exit is overwritten by the solution matrix
 X.

LDB - INTEGER.
 On entry, LDB specifies the first dimension of B as
 declared in the calling (sub) program. LDB
 must be at least $\max(1, m)$. Unchanged on exit.

— dtrsm.f —

```

SUBROUTINE DTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
$                B, LDB )
*   .. Scalar Arguments ..
CHARACTER*1      SIDE, UPLO, TRANSA, DIAG
INTEGER          M, N, LDA, LDB
DOUBLE PRECISION ALPHA
*   .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), B( LDB, * )
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*   .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
*   .. External Subroutines ..
EXTERNAL         XERBLA
*   .. Intrinsic Functions ..
INTRINSIC        MAX
*   .. Local Scalars ..
LOGICAL          LSIDE, NOUNIT, UPPER
INTEGER          I, INFO, J, K, NROWA
DOUBLE PRECISION TEMP
*   .. Parameters ..
DOUBLE PRECISION ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
LSIDE = LSAME( SIDE , 'L' )
IF( LSIDE )THEN
    NROWA = M
ELSE
    NROWA = N
END IF
NOUNIT = LSAME( DIAG , 'N' )
UPPER = LSAME( UPLO , 'U' )

```

```

*
      INFO = 0
      IF( ( .NOT.LSIDE ) .AND.
$      ( .NOT.LSAME( SIDE , 'R' ) ) ) THEN
          INFO = 1
      ELSE IF( ( .NOT.UPPER ) .AND.
$      ( .NOT.LSAME( UPLO , 'L' ) ) ) THEN
          INFO = 2
      ELSE IF( ( .NOT.LSAME( TRANSA, 'N' ) ) .AND.
$      ( .NOT.LSAME( TRANSA, 'T' ) ) .AND.
$      ( .NOT.LSAME( TRANSA, 'C' ) ) ) THEN
          INFO = 3
      ELSE IF( ( .NOT.LSAME( DIAG , 'U' ) ) .AND.
$      ( .NOT.LSAME( DIAG , 'N' ) ) ) THEN
          INFO = 4
      ELSE IF( M .LT.0 ) THEN
          INFO = 5
      ELSE IF( N .LT.0 ) THEN
          INFO = 6
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
          INFO = 9
      ELSE IF( LDB.LT.MAX( 1, M ) ) THEN
          INFO = 11
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DTRSM ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
*      And when alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
          DO 20, J = 1, N
              DO 10, I = 1, M
                  B( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
          RETURN
      END IF
*
*      Start the operations.
*
      IF( LSIDE ) THEN
          IF( LSAME( TRANSA, 'N' ) ) THEN
*

```

```

*      Form B := alpha*inv( A )*B.
*
      IF( UPPER )THEN
        DO 60, J = 1, N
          IF( ALPHA.NE.ONE )THEN
            DO 30, I = 1, M
              B( I, J ) = ALPHA*B( I, J )
30             CONTINUE
            END IF
            DO 50, K = M, 1, -1
              IF( B( K, J ).NE.ZERO )THEN
                IF( NOUNIT )
                  $      B( K, J ) = B( K, J )/A( K, K )
                DO 40, I = 1, K - 1
                  B( I, J ) = B( I, J ) - B( K, J )*A( I, K )
40                 CONTINUE
                END IF
              CONTINUE
            END IF
          CONTINUE
        CONTINUE
      ELSE
        DO 100, J = 1, N
          IF( ALPHA.NE.ONE )THEN
            DO 70, I = 1, M
              B( I, J ) = ALPHA*B( I, J )
70             CONTINUE
            END IF
            DO 90 K = 1, M
              IF( B( K, J ).NE.ZERO )THEN
                IF( NOUNIT )
                  $      B( K, J ) = B( K, J )/A( K, K )
                DO 80, I = K + 1, M
                  B( I, J ) = B( I, J ) - B( K, J )*A( I, K )
80                 CONTINUE
                END IF
              CONTINUE
            END IF
          CONTINUE
        CONTINUE
      END IF
    ELSE
      *
      *      Form B := alpha*inv( A' )*B.
      *
      IF( UPPER )THEN
        DO 130, J = 1, N
          DO 120, I = 1, M
            TEMP = ALPHA*B( I, J )
            DO 110, K = 1, I - 1
              TEMP = TEMP - A( K, I )*B( K, J )
110             CONTINUE
            IF( NOUNIT )
              $      TEMP = TEMP/A( I, I )

```

```

        B( I, J ) = TEMP
120      CONTINUE
130      CONTINUE
      ELSE
        DO 160, J = 1, N
          DO 150, I = M, 1, -1
            TEMP = ALPHA*B( I, J )
            DO 140, K = I + 1, M
              TEMP = TEMP - A( K, I )*B( K, J )
140          CONTINUE
            IF( NOUNIT )
              $      TEMP = TEMP/A( I, I )
              B( I, J ) = TEMP
150          CONTINUE
160        CONTINUE
      END IF
    END IF
  ELSE
    IF( LSAME( TRANSA, 'N' ) )THEN
*
*      Form B := alpha*B*inv( A ).
*
      IF( UPPER )THEN
        DO 210, J = 1, N
          IF( ALPHA.NE.ONE )THEN
            DO 170, I = 1, M
              B( I, J ) = ALPHA*B( I, J )
170          CONTINUE
            END IF
            DO 190, K = 1, J - 1
              IF( A( K, J ).NE.ZERO )THEN
                DO 180, I = 1, M
                  B( I, J ) = B( I, J ) - A( K, J )*B( I, K )
180              CONTINUE
            END IF
          CONTINUE
190        CONTINUE
        IF( NOUNIT )THEN
          TEMP = ONE/A( J, J )
          DO 200, I = 1, M
            B( I, J ) = TEMP*B( I, J )
200          CONTINUE
        END IF
210      CONTINUE
    ELSE
      DO 260, J = N, 1, -1
        IF( ALPHA.NE.ONE )THEN
          DO 220, I = 1, M
            B( I, J ) = ALPHA*B( I, J )
220          CONTINUE
        END IF

```

```

DO 240, K = J + 1, N
  IF( A( K, J ).NE.ZERO )THEN
    DO 230, I = 1, M
      B( I, J ) = B( I, J ) - A( K, J )*B( I, K )
230    CONTINUE
    END IF
240    CONTINUE
    IF( NOUNIT )THEN
      TEMP = ONE/A( J, J )
      DO 250, I = 1, M
        B( I, J ) = TEMP*B( I, J )
250      CONTINUE
    END IF
260    CONTINUE
  END IF
ELSE
*
*   Form B := alpha*B*inv( A' ).
*
  IF( UPPER )THEN
    DO 310, K = N, 1, -1
      IF( NOUNIT )THEN
        TEMP = ONE/A( K, K )
        DO 270, I = 1, M
          B( I, K ) = TEMP*B( I, K )
270        CONTINUE
      END IF
      DO 290, J = 1, K - 1
        IF( A( J, K ).NE.ZERO )THEN
          TEMP = A( J, K )
          DO 280, I = 1, M
            B( I, J ) = B( I, J ) - TEMP*B( I, K )
280          CONTINUE
        END IF
      CONTINUE
290    CONTINUE
    IF( ALPHA.NE.ONE )THEN
      DO 300, I = 1, M
        B( I, K ) = ALPHA*B( I, K )
300      CONTINUE
    END IF
310    CONTINUE
  ELSE
    DO 360, K = 1, N
      IF( NOUNIT )THEN
        TEMP = ONE/A( K, K )
        DO 320, I = 1, M
          B( I, K ) = TEMP*B( I, K )
320        CONTINUE
      END IF
      DO 340, J = K + 1, N

```



```

        IF( A( J, K ).NE.ZERO )THEN
            TEMP = A( J, K )
            DO 330, I = 1, M
                B( I, J ) = B( I, J ) - TEMP*B( I, K )
330          CONTINUE
            END IF
340          CONTINUE
            IF( ALPHA.NE.ONE )THEN
                DO 350, I = 1, M
                    B( I, K ) = ALPHA*B( I, K )
350          CONTINUE
            END IF
360          CONTINUE
        END IF
    END IF
END IF
*
    RETURN
*
*   End of DTRSM .
*
    END

```

— BLAS 3 dtrsm —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dtrsm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array double-float (*)) b a)
              (type (double-float) alpha)
              (type fixnum ldb$ lda n m)
              (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (uplo character uplo-%data% uplo-%offset%)
       (transa character transa-%data% transa-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%))
      (prog ((temp 0.0) (i 0) (info 0) (j 0) (k 0) (nrowa 0) (lside nil)
              (nounit nil) (upper nil))
        (declare (type (double-float) temp)
                  (type fixnum i info j k nrowa)
                  (type (member t nil) lside nounit upper))
        (setf lside (char-equal side #\L))

```

```

(cond
  (lside
    (setf nrowa m))
  (t
    (setf nrowa n)))
(setf nunit (char-equal diag #\N))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
  ((and (not lside) (not (char-equal side #\R)))
    (setf info 1))
  ((and (not upper) (not (char-equal uplo #\L)))
    (setf info 2))
  ((and (not (char-equal transa #\N))
        (not (char-equal transa #\T))
        (not (char-equal transa #\C)))
    (setf info 3))
  ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
    (setf info 4))
  (< m 0)
    (setf info 5))
  (< n 0)
    (setf info 6))
  (< lda (max (the fixnum 1) (the fixnum nrowa)))
    (setf info 9))
  (< ldb$ (max (the fixnum 1) (the fixnum m)))
    (setf info 11)))
(cond
  (/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DTRSM" info)
    (go end_label)))
(if (= n 0) (go end_label))
(cond
  (= alpha zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *))
                          b-%offset%)
          zero))))))
  (go end_label)))
(cond
  (lside

```



```

((1 ldb$) (1 *))
b-%offset%)

(-
  (f2cl-lib:fref b-%data%
    (i j)
    ((1 ldb$) (1 *))
    b-%offset%)

  (*
    (f2cl-lib:fref b-%data%
      (k j)
      ((1 ldb$) (1 *))
      b-%offset%)
    (f2cl-lib:fref a-%data%
      (i k)
      ((1 lda) (1 *))
      a-%offset%)))))))))

(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)

            (* alpha
              (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))
                b-%offset%))))))
        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
          (> k m) nil)
        (tagbody
          (cond
            ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
              (if nunit
                (setf (f2cl-lib:fref b-%data%
                  (k j)
                  ((1 ldb$) (1 *))
                  b-%offset%)

                  (/
                    (f2cl-lib:fref b-%data%
                      (k j)
                      ((1 ldb$) (1 *))
                      b-%offset%)
                    (f2cl-lib:fref a-%data%

```

```

(k k)
((1 lda) (1 *))
a-%offset%)))
(f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
  (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref b-%data%
    (i j)
    ((1 ldb$) (1 *))
    b-%offset%))
    (-
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
      (*
        (f2cl-lib:fref b-%data%
          (k j)
          ((1 ldb$) (1 *))
          b-%offset%)
        (f2cl-lib:fref a-%data%
          (i k)
          ((1 lda) (1 *))
          a-%offset%)))))))))
(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
          (tagbody
            (setf temp
              (* alpha
                (f2cl-lib:fref b-%data%
                  (i j)
                  ((1 ldb$) (1 *))
                  b-%offset%)))
              (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                (> k
                  (f2cl-lib:int-add i
                    (f2cl-lib:int-sub
                      1)))
                  nil)
                (tagbody
                  (setf temp
                    (- temp
                      (*

```

```

(f2cl-lib:fref a-%data%
              (k i)
              ((1 lda) (1 *)))
a-%offset%)
(f2cl-lib:fref b-%data%
              (k j)
              ((1 ldb$) (1 *)))
b-%offset%))))))

(if nunit
  (setf temp
    (/ temp
      (f2cl-lib:fref a-%data%
                    (i i)
                    ((1 lda) (1 *)))
      a-%offset%))))

(setf (f2cl-lib:fref b-%data%
                    (i j)
                    ((1 ldb$) (1 *)))
      b-%offset%)

temp))))))

(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)

 (tagbody
  (f2cl-lib:fdo (i m
    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    (> i 1) nil)

  (tagbody
   (setf temp
     (* alpha
       (f2cl-lib:fref b-%data%
                     (i j)
                     ((1 ldb$) (1 *)))
       b-%offset%)))

   (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
     (f2cl-lib:int-add k 1))
     (> k m) nil)

   (tagbody
    (setf temp
      (- temp
        (*
          (f2cl-lib:fref a-%data%
                        (k i)
                        ((1 lda) (1 *)))
          a-%offset%)
          (f2cl-lib:fref b-%data%
                        (k j)
                        ((1 ldb$) (1 *)))
          b-%offset%))))))

  (if nunit
    (f2cl-lib:fref a-%data%
                  (k i)
                  ((1 lda) (1 *)))
    a-%offset%)
    (f2cl-lib:fref b-%data%
                  (k j)
                  ((1 ldb$) (1 *)))
    b-%offset%))))))

```

```

        (setf temp
          (/ temp
             (f2cl-lib:fref a-%data%
                           (i i)
                           ((1 lda) (1 *))
                           a-%offset%))))
        (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
              temp)))))))))
(t
 (cond
  ((char-equal transa #\N)
   (cond
    (upper
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (cond
       ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i m) nil)
        (tagbody
         (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
               (* alpha
                  (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%))))))
         (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                       ((> k
                          (f2cl-lib:int-add j
                           (f2cl-lib:int-sub 1)))
                        nil)
         (tagbody
          (cond
           ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i m) nil)
            (tagbody
             (setf (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)
                   (-
                     (f2cl-lib:fref b-%data%

```

```

                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
(*
  (f2cl-lib:fref a-%data%
    (k j)
    ((1 lda) (1 *))
    a-%offset%)
  (f2cl-lib:fref b-%data%
    (i k)
    ((1 ldb$) (1 *))
    b-%offset%)))))))))
(cond
  (nunit
    (setf temp
      (/ one
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
        (* temp
          (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)))))))))
(t
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)
  (tagbody
    (cond
      ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)
            (* alpha
              (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))
                b-%offset%)))))))))

```



```

(f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
  (> k 1) nil)
(tagbody
  (cond
    (nounit
      (setf temp
        (/ one
          (f2cl-lib:fref a-%data%
            (k k)
            ((1 lda) (1 *))
            a-%offset%)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref b-%data%
          (i k)
          ((1 ldb$) (1 *))
          b-%offset%)
          (* temp
            (f2cl-lib:fref b-%data%
              (i k)
              ((1 ldb$) (1 *))
              b-%offset%))))))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j
          (f2cl-lib:int-add k
            (f2cl-lib:int-sub 1)))
          nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
            (setf temp
              (f2cl-lib:fref a-%data%
                (j k)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))
                b-%offset%)
                (-
                  (f2cl-lib:fref b-%data%
                    (i j)
                    ((1 ldb$) (1 *))
                    b-%offset%)
                  (* temp
                    (f2cl-lib:fref b-%data%

```

```

                                (i k)
                                ((1 ldb$) (1 *))
                                b-%offset%)))))))))
(cond
  ((/= alpha one)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                 (> i m) nil)
   (tagbody
    (setf (f2cl-lib:fref b-%data%
                        (i k)
                        ((1 ldb$) (1 *))
                        b-%offset%))
          (* alpha
             (f2cl-lib:fref b-%data%
                             (i k)
                             ((1 ldb$) (1 *))
                             b-%offset%)))))))))
(t
 (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
               (> k n) nil)
 (tagbody
  (cond
   (nounit
    (setf temp
          (/ one
             (f2cl-lib:fref a-%data%
                             (k k)
                             ((1 lda) (1 *))
                             a-%offset%)))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  (> i m) nil)
    (tagbody
     (setf (f2cl-lib:fref b-%data%
                         (i k)
                         ((1 ldb$) (1 *))
                         b-%offset%))
           (* temp
              (f2cl-lib:fref b-%data%
                              (i k)
                              ((1 ldb$) (1 *))
                              b-%offset%))))))
    (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
                  (f2cl-lib:int-add j 1))
                  (> j n) nil)
    (tagbody
     (cond
      ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
       (setf temp
             (f2cl-lib:fref a-%data%
                             (j k)

```

```

((1 lda) (1 *))
a-%offset%)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref b-%data%
    (i j)
    ((1 ldb$) (1 *))
    b-%offset%)
    (-
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
      (* temp
        (f2cl-lib:fref b-%data%
          (i k)
          ((1 ldb$) (1 *))
          b-%offset%)))))))))
(cond
  ((/= alpha one)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
        (i k)
        ((1 ldb$) (1 *))
        b-%offset%)
        (* alpha
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%)))))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))

```

zgemm BLAS

— zgemm.input —

```

)set break resume
)sys rm -f zgemm.output
)spool zgemm.output
)set message test on

```

```
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— zgemm.help —

```
=====
zgemm examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZGEMM - perform one of the matrix-matrix operations $C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

SYNOPSIS

SUBROUTINE ZGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
B, LDB, BETA, C, LDC)

CHARACTER*1 TRANSA, TRANSB

INTEGER M, N, K, LDA, LDB, LDC

COMPLEX*16 ALPHA, BETA

COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE

ZGEMM performs one of the matrix-matrix operations

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X'$ or $\text{op}(X) = \text{conjg}(X')$,

alpha and beta are scalars, and A, B and C are matrices,
with $\text{op}(A)$ an m by k matrix, $\text{op}(B)$ a k by n matrix
and C an m by n matrix.

PARAMETERS

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form
of $\text{op}(A)$ to be used in the matrix multiplication as fol-

lows:

TRANSA = 'N' or 'n', op(A) = A.

TRANSA = 'T' or 't', op(A) = A'.

TRANSA = 'C' or 'c', op(A) = conjg(A').

Unchanged on exit.

TRANSB - CHARACTER*1. On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B'.

TRANSB = 'C' or 'c', op(B) = conjg(B').

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

-
COMPLEX*16 array of DIMENSION (LDA, ka), where k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array

A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least $\max(1, m)$, otherwise LDA must be at least $\max(1, k)$. Unchanged on exit.

kb is

B

-
COMPLEX*16 array of DIMENSION (LDB, kb), where n when TRANS = 'N' or 'n', and is k otherwise. Before entry with TRANS = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least $\max(1, k)$, otherwise LDB must be at least $\max(1, n)$. Unchanged on exit.

BETA - COMPLEX*16 .

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix $(\alpha * \text{op}(A) * \text{op}(B) + \beta * C)$.

LDC - INTEGER.

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, m)$. Unchanged on exit.

— zgemm.f —

SUBROUTINE ZGEMM (TRANS, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,

```

$          BETA, C, LDC )
*      .. Scalar Arguments ..
      CHARACTER*1      TRANSA, TRANSB
      INTEGER           M, N, K, LDA, LDB, LDC
      COMPLEX*16        ALPHA, BETA
*      .. Array Arguments ..
      COMPLEX*16        A( LDA, * ), B( LDB, * ), C( LDC, * )
*
*
*      Level 3 Blas routine.
*
*      -- Written on 8-February-1989.
*      Jack Dongarra, Argonne National Laboratory.
*      Iain Duff, AERE Harwell.
*      Jeremy Du Croz, Numerical Algorithms Group Ltd.
*      Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*      .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL          LSAME
*      .. External Subroutines ..
      EXTERNAL          XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC          DCONJG, MAX
*      .. Local Scalars ..
      LOGICAL           CONJA, CONJB, NOTA, NOTB
      INTEGER           I, INFO, J, L, NCOLA, NROWA, NROWB
      COMPLEX*16        TEMP
*      .. Parameters ..
      COMPLEX*16        ONE
      PARAMETER          ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16        ZERO
      PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Executable Statements ..
*
*
*      Set  NOTA  and  NOTB  as  true  if  A  and  B  respectively are not
*      conjugated or transposed, set  CONJA  and  CONJB  as true if  A  and
*      B  respectively are to be transposed but not conjugated and set
*      NROWA, NCOLA and NROWB as the number of rows and columns of  A
*      and the number of rows of  B  respectively.
*
*
      NOTA = LSAME( TRANSA, 'N' )
      NOTB = LSAME( TRANSB, 'N' )
      CONJA = LSAME( TRANSA, 'C' )
      CONJB = LSAME( TRANSB, 'C' )
      IF( NOTA )THEN
          NROWA = M
          NCOLA = K

```



```

ELSE
    NROWA = K
    NCOLA = M
END IF
IF( NOTB )THEN
    NROWB = K
ELSE
    NROWB = N
END IF
*
*   Test the input parameters.
*
    INFO = 0
    IF(      ( .NOT.NOTA                ) .AND.
$      ( .NOT.CONJA                ) .AND.
$      ( .NOT.LSAME( TRANSA, 'T' ) )      ) THEN
        INFO = 1
    ELSE IF( ( .NOT.NOTB                ) .AND.
$      ( .NOT.CONJB                ) .AND.
$      ( .NOT.LSAME( TRANSB, 'T' ) )      ) THEN
        INFO = 2
    ELSE IF( M .LT.0                ) THEN
        INFO = 3
    ELSE IF( N .LT.0                ) THEN
        INFO = 4
    ELSE IF( K .LT.0                ) THEN
        INFO = 5
    ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
        INFO = 8
    ELSE IF( LDB.LT.MAX( 1, NROWB ) ) THEN
        INFO = 10
    ELSE IF( LDC.LT.MAX( 1, M      ) ) THEN
        INFO = 13
    END IF
    IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZGEMM ', INFO )
        RETURN
    END IF
*
*   Quick return if possible.
*
    IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$      ( ( ( ALPHA.EQ.ZERO ) .OR.( K.EQ.0 ) ) .AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   And when  alpha.eq.zero.
*
    IF( ALPHA.EQ.ZERO )THEN
        IF( BETA.EQ.ZERO )THEN
            DO 20, J = 1, N

```

```

DO 10, I = 1, M
  C( I, J ) = ZERO
10  CONTINUE
20  CONTINUE
    ELSE
      DO 40, J = 1, N
        DO 30, I = 1, M
          C( I, J ) = BETA*C( I, J )
30      CONTINUE
40      CONTINUE
        END IF
      RETURN
    END IF

*
*  Start the operations.
*
  IF( NOTB )THEN
    IF( NOTA )THEN
*
*      Form C := alpha*A*B + beta*C.
*
      DO 90, J = 1, N
        IF( BETA.EQ.ZERO )THEN
          DO 50, I = 1, M
            C( I, J ) = ZERO
50          CONTINUE
        ELSE IF( BETA.NE.ONE )THEN
          DO 60, I = 1, M
            C( I, J ) = BETA*C( I, J )
60          CONTINUE
        END IF
        DO 80, L = 1, K
          IF( B( L, J ).NE.ZERO )THEN
            TEMP = ALPHA*B( L, J )
            DO 70, I = 1, M
              C( I, J ) = C( I, J ) + TEMP*A( I, L )
70          CONTINUE
            END IF
          CONTINUE
60          CONTINUE
90          CONTINUE
        ELSE IF( CONJA )THEN
*
*      Form C := alpha*conjg( A' )*B + beta*C.
*
      DO 120, J = 1, N
        DO 110, I = 1, M
          TEMP = ZERO
          DO 100, L = 1, K
            TEMP = TEMP + DCONJG( A( L, I ) )*B( L, J )
100         CONTINUE

```

```

        IF( BETA.EQ.ZERO )THEN
            C( I, J ) = ALPHA*TEMP
        ELSE
            C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
        END IF
110      CONTINUE
120      CONTINUE
    ELSE
*
*      Form C := alpha*A'*B + beta*C
*
        DO 150, J = 1, N
            DO 140, I = 1, M
                TEMP = ZERO
                DO 130, L = 1, K
                    TEMP = TEMP + A( L, I )*B( L, J )
130              CONTINUE
                IF( BETA.EQ.ZERO )THEN
                    C( I, J ) = ALPHA*TEMP
                ELSE
                    C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                END IF
140              CONTINUE
150            CONTINUE
        END IF
    ELSE IF( NOTA )THEN
        IF( CONJB )THEN
*
*      Form C := alpha*A*conjg( B' ) + beta*C.
*
            DO 200, J = 1, N
                IF( BETA.EQ.ZERO )THEN
                    DO 160, I = 1, M
                        C( I, J ) = ZERO
160                  CONTINUE
                ELSE IF( BETA.NE.ONE )THEN
                    DO 170, I = 1, M
                        C( I, J ) = BETA*C( I, J )
170                  CONTINUE
                END IF
                DO 190, L = 1, K
                    IF( B( J, L ).NE.ZERO )THEN
                        TEMP = ALPHA*DCONJG( B( J, L ) )
                        DO 180, I = 1, M
                            C( I, J ) = C( I, J ) + TEMP*A( I, L )
180                      CONTINUE
                        END IF
190                  CONTINUE
200                CONTINUE
            ELSE

```

```

*
*      Form C := alpha*A*B'          + beta*C
*
      DO 250, J = 1, N
        IF( BETA.EQ.ZERO )THEN
          DO 210, I = 1, M
            C( I, J ) = ZERO
210          CONTINUE
        ELSE IF( BETA.NE.ONE )THEN
          DO 220, I = 1, M
            C( I, J ) = BETA*C( I, J )
220          CONTINUE
        END IF
        DO 240, L = 1, K
          IF( B( J, L ).NE.ZERO )THEN
            TEMP = ALPHA*B( J, L )
            DO 230, I = 1, M
              C( I, J ) = C( I, J ) + TEMP*A( I, L )
230            CONTINUE
          END IF
        CONTINUE
240      CONTINUE
250    CONTINUE
      END IF
      ELSE IF( CONJA )THEN
        IF( CONJB )THEN
*
*      Form C := alpha*conjg( A' )*conjg( B' ) + beta*C.
*
          DO 280, J = 1, N
            DO 270, I = 1, M
              TEMP = ZERO
              DO 260, L = 1, K
                TEMP = TEMP +
$                DCONJG( A( L, I ) )*DCONJG( B( J, L ) )
260              CONTINUE
              IF( BETA.EQ.ZERO )THEN
                C( I, J ) = ALPHA*TEMP
              ELSE
                C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
              END IF
            CONTINUE
270          CONTINUE
280        CONTINUE
      ELSE
*
*      Form C := alpha*conjg( A' )*B' + beta*C
*
          DO 310, J = 1, N
            DO 300, I = 1, M
              TEMP = ZERO
              DO 290, L = 1, K

```

```

                TEMP = TEMP + DCONJG( A( L, I ) ) * B( J, L )
290            CONTINUE
                IF( BETA.EQ.ZERO ) THEN
                    C( I, J ) = ALPHA * TEMP
                ELSE
                    C( I, J ) = ALPHA * TEMP + BETA * C( I, J )
                END IF
300            CONTINUE
310        CONTINUE
        END IF
    ELSE
        IF( CONJB ) THEN
*
*           Form C := alpha*A'*conjg( B' ) + beta*C
*
            DO 340, J = 1, N
                DO 330, I = 1, M
                    TEMP = ZERO
                    DO 320, L = 1, K
                        TEMP = TEMP + A( L, I ) * DCONJG( B( J, L ) )
320                CONTINUE
                    IF( BETA.EQ.ZERO ) THEN
                        C( I, J ) = ALPHA * TEMP
                    ELSE
                        C( I, J ) = ALPHA * TEMP + BETA * C( I, J )
                    END IF
330                CONTINUE
340            CONTINUE
        ELSE
*
*           Form C := alpha*A'*B' + beta*C
*
            DO 370, J = 1, N
                DO 360, I = 1, M
                    TEMP = ZERO
                    DO 350, L = 1, K
                        TEMP = TEMP + A( L, I ) * B( J, L )
350                CONTINUE
                    IF( BETA.EQ.ZERO ) THEN
                        C( I, J ) = ALPHA * TEMP
                    ELSE
                        C( I, J ) = ALPHA * TEMP + BETA * C( I, J )
                    END IF
360                CONTINUE
370            CONTINUE
        END IF
    END IF
*
    RETURN
*
```

```

*      End of ZGEMM .
*
      END

```

— BLAS 3 zgemm —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zgemm (transa transb m n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
      (type (complex double-float) beta alpha)
      (type fixnum ldc ldb$ lda k n m)
      (type character transb transa))
    (f2cl-lib:with-multi-array-data
      ((transa character transa-%data% transa-%offset%)
       (transb character transb-%data% transb-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (b (complex double-float) b-%data% b-%offset%)
       (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0) (ncola 0) (nrowa 0)
        (nrowb 0) (conja nil) (conjb nil) (nota nil) (notb nil))
        (declare (type (complex double-float) temp)
          (type fixnum i info j l ncola nrowa nrowb)
          (type (member t nil) conja conjb nota notb))
        (setf nota (char-equal transa #\N))
        (setf notb (char-equal transb #\N))
        (setf conja (char-equal transa #\C))
        (setf conjb (char-equal transb #\C))
        (cond
          (nota
            (setf nrowa m)
            (setf ncola k))
          (t
            (setf nrowa k)
            (setf ncola m)))
        (cond
          (notb
            (setf nrowb k))
          (t
            (setf nrowb n)))
        (setf info 0)
        (cond
          ((and (not nota) (not conja) (not (char-equal transa #\T)))
            (setf info 1))
          ((and (not notb) (not conjb) (not (char-equal transb #\T)))
            (setf info 2))

```

```

((< m 0)
 (setf info 3))
((< n 0)
 (setf info 4))
((< k 0)
 (setf info 5))
((< lda (max (the fixnum 1) (the fixnum nrowa)))
 (setf info 8))
((< ldb$
 (max (the fixnum 1) (the fixnum nrowb)))
 (setf info 10))
((< ldc (max (the fixnum 1) (the fixnum m)))
 (setf info 13))
(cond
 ((/= info 0)
 (error
  " ** On entry to ~a parameter number ~a had an illegal value~%"
  "ZGEMM" info)
 (go end_label)))
(if (or (= m 0) (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
 (go end_label))
(cond
 ((= alpha zero)
 (cond
 ((= beta zero)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j n) nil)
 (tagbody
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i m) nil)
 (tagbody
 (setf (f2cl-lib:fref c-%data%
 (i j)
 ((1 ldc) (1 *))
 c-%offset%)
 zero))))))
 (t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j n) nil)
 (tagbody
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i m) nil)
 (tagbody
 (setf (f2cl-lib:fref c-%data%
 (i j)
 ((1 ldc) (1 *))
 c-%offset%)
 (* beta
 (f2cl-lib:fref c-%data%
 (i j)

```

```

((1 ldc) (1 *))
c-%offset%)))))))))
(go end_label)))
(cond
(notb
(cond
(nota
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(cond
(= beta zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i m) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%
zero))))
(/= beta one)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i m) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%
(* beta
(f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
(> l k) nil)
(tagbody
(cond
(/= (f2cl-lib:fref b (l j) ((1 ldb$) (1 *))) zero)
(setf temp
(* alpha
(f2cl-lib:fref b-%data%
(l j)
((1 ldb$) (1 *))
b-%offset%)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i m) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))

```



```

                                c-%offset%)
(+
  (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* temp
    (f2cl-lib:fref a-%data%
      (i 1)
      ((1 lda) (1 *))
      a-%offset%)))))))))
(conja
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf temp zero)
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
        (> l k) nil)
        (tagbody
          (setf temp
            (+ temp
              (*
                (f2cl-lib:dconjg
                  (f2cl-lib:fref a-%data%
                    (l i)
                    ((1 lda) (1 *))
                    a-%offset%))
                (f2cl-lib:fref b-%data%
                  (l j)
                  ((1 ldb) (1 *))
                  b-%offset%))))))
          (cond
            ((= beta zero)
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (* alpha temp)))
            (t
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (+ (* alpha temp)
                  (* beta
                    (f2cl-lib:fref c-%data%
                      (i j)

```

```

((1 ldc) (1 *))
c-%offset%)))))))))

(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf temp zero)
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
        (> l k) nil)
        (tagbody
          (setf temp
            (+ temp
              (*
                (f2cl-lib:fref a-%data%
                  (l i)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref b-%data%
                  (l j)
                  ((1 ldb) (1 *))
                  b-%offset%))))))
          (cond
            ((= beta zero)
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (* alpha temp)))
            (t
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (+ (* alpha temp)
                  (* beta
                    (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *))
                      c-%offset%))))))
          (nota
            (cond
              (conjb
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  (> j n) nil)
                (tagbody
                  (cond
                    ((= beta zero)

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
      zero))))
( /= beta one)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
      (* beta
        (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
  ((> l k) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref b-%data%
                              (j 1)
                              ((1 ldb$) (1 *))
                              b-%offset%))))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                (+
                  (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                  (* temp
                    (f2cl-lib:fref a-%data%
                              (i 1)
                              ((1 lda) (1 *))
                              a-%offset%))))))))))

```

```
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)

(tagbody
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)

      (tagbody
        (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              zero))))

    ((/= beta one)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)

      (tagbody
        (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              (* beta
                (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%))))))

    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
      (> l k) nil)

    (tagbody
      (cond
        ((/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero)
          (setf temp
            (* alpha
              (f2cl-lib:fref b-%data%
                              (j 1)
                              ((1 ldb$) (1 *))
                              b-%offset%)))

          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i m) nil)

          (tagbody
            (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                (+
                  (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
```



```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf temp zero)
    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
      (> l k) nil)
      (tagbody
        (setf temp
          (+ temp
            (*
              (f2cl-lib:dconjg
                (f2cl-lib:fref a-%data%
                  (l i)
                  ((1 lda) (1 *))
                  a-%offset%))
              (f2cl-lib:fref b-%data%
                (j l)
                ((1 ldb$) (1 *))
                b-%offset%))))))
        (cond
          ((= beta zero)
            (setf (f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *))
              c-%offset%)
              (* alpha temp)))
          (t
            (setf (f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *))
              c-%offset%)
              (+ (* alpha temp)
                (* beta
                  (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%))))))))))
    (t
      (cond
        (conjb
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            (> j n) nil)
          (tagbody
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i m) nil)
            (tagbody
              (setf temp zero)

```

```

(f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
  (tagbody
    (setf temp
      (+ temp
        (*
          (f2cl-lib:fref a-%data%
            (l i)
            ((1 lda) (1 *))
            a-%offset%)
          (f2cl-lib:dconjg
            (f2cl-lib:fref b-%data%
              (j 1)
              ((1 ldb$) (1 *))
              b-%offset%))))))
    (cond
      ((= beta zero)
        (setf (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)
          (* alpha temp)))
      (t
        (setf (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)
          (+ (* alpha temp)
            (* beta
              (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%))))))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
          (tagbody
            (setf temp zero)
            (f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
              (> 1 k) nil)
              (tagbody
                (setf temp
                  (+ temp
                    (*
                      (f2cl-lib:fref a-%data%
                        (l i)
                        ((1 lda) (1 *))

```

```

                                a-%offset%)
(f2cl-lib:fref b-%data%
  (j 1)
  ((1 ldb$) (1 *)))
b-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *)))
      c-%offset%)
      (* alpha temp)))
(t
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *)))
    c-%offset%)
    (+ (* alpha temp)
      (* beta
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *)))
          c-%offset%)))))))))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil))))

```

zhemm BLAS

— zhemm.input —

```

)set break resume
)sys rm -f zhemm.output
)spool zhemm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhemm.help —


```
=====
zhemm examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZHEMM - perform one of the matrix-matrix operations $C := \alpha A * B + \beta C$,

SYNOPSIS

SUBROUTINE ZHEMM (SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

CHARACTER*1 SIDE, UPLO

INTEGER M, N, LDA, LDB, LDC

COMPLEX*16 ALPHA, BETA

COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE

ZHEMM performs one of the matrix-matrix operations

or

$C := \alpha B * A + \beta C$,

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether the hermitian matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l' $C := \alpha A * B + \beta C$,

SIDE = 'R' or 'r' $C := \alpha B * A + \beta C$,

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the hermitian matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the hermitian matrix is to be referenced.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix C. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix C. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

-
COMPLEX*16 array of DIMENSION (LDA, ka), where m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced. Note that the imaginary parts of the diagonal elements

need not be set, they are assumed to be zero.
Unchanged on exit.

LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least $\max(1, m)$, otherwise LDA must be at least $\max(1, n)$.
Unchanged on exit.

B - COMPLEX*16 array of DIMENSION (LDB, n).

Before entry, the leading m by n part of the array B must contain the matrix B. Unchanged on exit.

LDB - INTEGER.
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least $\max(1, m)$. Unchanged on exit.

BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n).
Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.

LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, m)$. Unchanged on exit.

— zhemm.f —

```

SUBROUTINE ZHEMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
$               BETA, C, LDC )
* .. Scalar Arguments ..
CHARACTER*1    SIDE, UPLO
INTEGER        M, N, LDA, LDB, LDC
COMPLEX*16     ALPHA, BETA
* .. Array Arguments ..
```

```

      COMPLEX*16      A( LDA, * ), B( LDB, * ), C( LDC, * )
      ..
*
*
* Level 3 Blas routine.
*
* -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
* .. External Functions ..
      LOGICAL      LSAME
      EXTERNAL     LSAME
* .. External Subroutines ..
      EXTERNAL     XERBLA
* .. Intrinsic Functions ..
      INTRINSIC    DCONJG, MAX, DBLE
* .. Local Scalars ..
      LOGICAL      UPPER
      INTEGER      I, INFO, J, K, NROWA
      COMPLEX*16   TEMP1, TEMP2
* .. Parameters ..
      COMPLEX*16   ONE
      PARAMETER    ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16   ZERO
      PARAMETER    ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* ..
* .. Executable Statements ..
*
* Set NROWA as the number of rows of A.
*
      IF( LSAME( SIDE, 'L' ) )THEN
        NROWA = M
      ELSE
        NROWA = N
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
* Test the input parameters.
*
      INFO = 0
      IF(      ( .NOT.LSAME( SIDE, 'L' ) ).AND.
$          ( .NOT.LSAME( SIDE, 'R' ) ) )THEN
        INFO = 1
      ELSE IF( ( .NOT.UPPER ) )AND.
$          ( .NOT.LSAME( UPLO, 'L' ) ) )THEN
        INFO = 2
      ELSE IF( M .LT.0 )THEN

```

```

        INFO = 3
      ELSE IF( N .LT.0 ) THEN
        INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
        INFO = 7
      ELSE IF( LDB.LT.MAX( 1, M ) ) THEN
        INFO = 9
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
        INFO = 12
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZHEMM ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$      ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
        IF( BETA.EQ.ZERO ) THEN
          DO 20, J = 1, N
            DO 10, I = 1, M
              C( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
        ELSE
          DO 40, J = 1, N
            DO 30, I = 1, M
              C( I, J ) = BETA*C( I, J )
30          CONTINUE
40          CONTINUE
        END IF
        RETURN
      END IF
*
*   Start the operations.
*
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*       Form  C := alpha*A*B + beta*C.
*
        IF( UPPER ) THEN
          DO 70, J = 1, N
            DO 60, I = 1, M
              TEMP1 = ALPHA*B( I, J )

```

```

      TEMP2 = ZERO
      DO 50, K = 1, I - 1
        C( K, J ) = C( K, J ) + TEMP1*A( K, I )
        TEMP2      = TEMP2      +
$              B( K, J )*DCONJG( A( K, I ) )
50      CONTINUE
      IF( BETA.EQ.ZERO )THEN
        C( I, J ) = TEMP1*DBLE( A( I, I ) ) +
$              ALPHA*TEMP2
      ELSE
        C( I, J ) = BETA *C( I, J )          +
$              TEMP1*DBLE( A( I, I ) ) +
$              ALPHA*TEMP2
      END IF
60      CONTINUE
70      CONTINUE
    ELSE
      DO 100, J = 1, N
        DO 90, I = M, 1, -1
          TEMP1 = ALPHA*B( I, J )
          TEMP2 = ZERO
          DO 80, K = I + 1, M
            C( K, J ) = C( K, J ) + TEMP1*A( K, I )
            TEMP2      = TEMP2      +
$              B( K, J )*DCONJG( A( K, I ) )
80      CONTINUE
          IF( BETA.EQ.ZERO )THEN
            C( I, J ) = TEMP1*DBLE( A( I, I ) ) +
$              ALPHA*TEMP2
          ELSE
            C( I, J ) = BETA *C( I, J )          +
$              TEMP1*DBLE( A( I, I ) ) +
$              ALPHA*TEMP2
          END IF
90      CONTINUE
100     CONTINUE
      END IF
    ELSE
*
*      Form C := alpha*B*A + beta*C.
*
      DO 170, J = 1, N
        TEMP1 = ALPHA*DBLE( A( J, J ) )
        IF( BETA.EQ.ZERO )THEN
          DO 110, I = 1, M
            C( I, J ) = TEMP1*B( I, J )
110     CONTINUE
          ELSE
            DO 120, I = 1, M
              C( I, J ) = BETA*C( I, J ) + TEMP1*B( I, J )

```

```

120      CONTINUE
      END IF
      DO 140, K = 1, J - 1
        IF( UPPER )THEN
          TEMP1 = ALPHA*A( K, J )
        ELSE
          TEMP1 = ALPHA*DCONJG( A( J, K ) )
        END IF
        DO 130, I = 1, M
          C( I, J ) = C( I, J ) + TEMP1*B( I, K )
130      CONTINUE
140      CONTINUE
      DO 160, K = J + 1, N
        IF( UPPER )THEN
          TEMP1 = ALPHA*DCONJG( A( J, K ) )
        ELSE
          TEMP1 = ALPHA*A( K, J )
        END IF
        DO 150, I = 1, M
          C( I, J ) = C( I, J ) + TEMP1*B( I, K )
150      CONTINUE
160      CONTINUE
170      CONTINUE
      END IF
*
      RETURN
*
*      End of ZHEMM .
*
      END

```

— BLAS 3 zhemm —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
(declare (type (complex double-float) one) (type (complex double-float) zero))
(defun zhemm (side uplo m n alpha a lda b ldb$ beta c ldc)
  (declare (type (simple-array (complex double-float) (*)) c b a)
    (type (complex double-float) beta alpha)
    (type fixnum ldc ldb$ lda n m)
    (type character uplo side))
  (f2cl-lib:with-multi-array-data
    ((side character side-%data% side-%offset%)
     (uplo character uplo-%data% uplo-%offset%)
     (a (complex double-float) a-%data% a-%offset%)
     (b (complex double-float) b-%data% b-%offset%)
     (c (complex double-float) c-%data% c-%offset%))

```

```

(prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0)
      (nrowa 0) (upper nil))
(declare (type (complex double-float) temp1 temp2)
          (type fixnum i info j k nrowa)
          (type (member t nil) upper))
(cond
  ((char-equal side #\L)
   (setf nrowa m))
  (t
   (setf nrowa n)))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
  ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
   (setf info 1))
  ((and (not upper) (not (char-equal uplo #\L)))
   (setf info 2))
  ((< m 0)
   (setf info 3))
  ((< n 0)
   (setf info 4))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 7))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 9))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 12)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZHEMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
    (go end_label))
(cond
  ((= alpha zero)
   (cond
     ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
                zero))))))

```



```

(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%))
        (* beta
          (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
  (go end_label)))
(cond
  ((char-equal side #\L)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i m) nil)
            (tagbody
              (setf temp1
                (* alpha
                  (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)))
                (setf temp2 zero)
                (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  (> k
                    (f2cl-lib:int-add i
                      (f2cl-lib:int-sub
                        1)))
                    nil)
                (tagbody
                  (setf (f2cl-lib:fref c-%data%
                                      (k j)
                                      ((1 ldc) (1 *))
                                      c-%offset%))
                    (+
                      (f2cl-lib:fref c-%data%
                                    (k j)
                                    ((1 ldc) (1 *))
                                    c-%offset%))

```

```

(* temp1
  (f2cl-lib:fref a-%data%
                (k i)
                ((1 lda) (1 *))
                a-%offset%)))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:fref b-%data%
                    (k j)
                    ((1 ldb$) (1 *))
                    b-%offset%)
      (f2cl-lib:dconjg
        (f2cl-lib:fref a-%data%
                      (k i)
                      ((1 lda) (1 *))
                      a-%offset%))))))
(cond
  ((= beta zero)
    (setf
      (f2cl-lib:fref c-%data% (i j) ((1 ldc) (1 *))
                    c-%offset%)
      (+
        (* temp1
          (coerce (realpart
                    (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *))
                    a-%offset%))
                    'double-float))
        (* alpha temp2))))
  (t
    (setf (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *))
                      c-%offset%)
      (+
        (* beta
          (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))
        (* temp1
          (coerce (realpart
                    (f2cl-lib:fref a-%data%
                      (i i)
                      ((1 lda) (1 *))
                      a-%offset%))
                    'double-float))
          (* alpha temp2))))))
  (t
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

        (<> j n) nil)
(tagbody
  (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    (<> i 1) nil)
  (tagbody
    (setf temp1
      (* alpha
        (f2cl-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *))
          b-%offset%)))
    (setf temp2 zero)
    (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
      (f2cl-lib:int-add k 1))
      (<> k m) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
        (k j)
        ((1 ldc) (1 *))
        c-%offset%)
        (+
          (f2cl-lib:fref c-%data%
            (k j)
            ((1 ldc) (1 *))
            c-%offset%)
          (* temp1
            (f2cl-lib:fref a-%data%
              (k i)
              ((1 lda) (1 *))
              a-%offset%))))))
    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:fref b-%data%
            (k j)
            ((1 ldb$) (1 *))
            b-%offset%)
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (k i)
              ((1 lda) (1 *))
              a-%offset%)))))))
    (cond
      ((= beta zero)
        (setf (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)
          (+
            (* temp1

```

```

(coerce (realpart
(f2cl-lib:fref a-%data%
(i i)
((1 lda) (1 *))
a-%offset%)) 'double-float))
(* alpha temp2))))
(t
(setf (f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%)
(
(+
(* beta
(f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%))
(* temp1
(coerce (realpart
(f2cl-lib:fref a-%data%
(i i)
((1 lda) (1 *))
a-%offset%)) 'double-float))
(* alpha temp2)))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j n) nil)
(tagbody
(setf temp1
(* alpha
(coerce (realpart
(f2cl-lib:fref a-%data%
(j j)
((1 lda) (1 *))
a-%offset%)) 'double-float)))
(cond
((= beta zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i m) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
(i j)
((1 ldc) (1 *))
c-%offset%)
(* temp1
(f2cl-lib:fref b-%data%
(i j)
((1 ldb$) (1 *))
b-%offset%))))))
(t

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%))
    (+
      (* beta
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%))
        (* temp1
          (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (cond
    (upper
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
            (k j)
            ((1 lda) (1 *))
            a-%offset%))))
    (t
      (setf temp1
        (* alpha
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (j k)
              ((1 lda) (1 *))
              a-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%))
    (+
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)

```

```

(* temp1
  (f2cl-lib:fref b-%data%
    (i k)
    ((1 ldb$) (1 *))
    b-%offset%))))))
(f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
  (> k n) nil)
(tagbody
  (cond
    (upper
      (setf temp1
        (* alpha
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
              (j k)
              ((1 lda) (1 *))
              a-%offset%))))))
    (t
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
            (k j)
            ((1 lda) (1 *))
            a-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
    (+
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)
      (* temp1
        (f2cl-lib:fref b-%data%
          (i k)
          ((1 ldb$) (1 *))
          b-%offset%))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil nil))))

```

zher2k BLAS**— zher2k.input —**

```

)set break resume
)sys rm -f zher2k.output
)spool zher2k.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zher2k.help —

```

=====
zher2k examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZHER2K - perform one of the hermitian rank 2k operations C
:= alpha*A*conjg(B') + conjg(alpha)*B*conjg(A') +
beta*C,

SYNOPSIS

SUBROUTINE ZHER2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
 BETA, C, LDC)

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDB, LDC

DOUBLE PRECISION BETA

COMPLEX*16 ALPHA

COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE

ZHER2K performs one of the hermitian rank 2k operations

or

```
C := alpha*conjg( A' )*B + conjg( alpha )*conjg( B' )*A +
beta*C,
```

where `alpha` and `beta` are scalars with `beta` real, `C` is an `n` by `n` hermitian matrix and `A` and `B` are `n` by `k` matrices in the first case and `k` by `n` matrices in the second case.

PARAMETERS

`UPLO` - CHARACTER*1.

On entry, `UPLO` specifies whether the upper or lower triangular part of the array `C` is to be referenced as follows:

`UPLO` = 'U' or 'u' Only the upper triangular part of `C` is to be referenced.

`UPLO` = 'L' or 'l' Only the lower triangular part of `C` is to be referenced.

Unchanged on exit.

`TRANS` - CHARACTER*1.

On entry, `TRANS` specifies the operation to be performed as follows:

`TRANS` = 'N' or 'n' `C` := `alpha`*`A`*`conjg(B')`

+ `conjg(alpha)`*`B`*`conjg(A')` + `beta`*`C`.

`TRANS` = 'C' or 'c' `C` := `alpha`*`conjg(A')`*`B`

+ `conjg(alpha)`*`conjg(B')`*`A` + `beta`*`C`.

Unchanged on exit.

`N` - INTEGER.

On entry, `N` specifies the order of the matrix `C`. `N` must be at least zero. Unchanged on exit.

`K` - INTEGER.

On entry with `TRANS` = 'N' or 'n', `K` specifies the number of columns of the matrices `A` and `B`, and on entry with `TRANS` = 'C' or 'c', `K` specifies the number of rows of the matrices `A` and `B`. `K` must be at least zero. Unchanged on exit.

`ALPHA` - COMPLEX*16 .

On entry, `ALPHA` specifies the scalar `alpha`.

Unchanged on exit.

ka is

A

-
COMPLEX*16 array of DIMENSION (LDA, ka), where
k when TRANS = 'N' or 'n', and is n otherwise.
Before entry with TRANS = 'N' or 'n', the leading
n by k part of the array A must contain the matrix
A, otherwise the leading k by n part of the array
A must contain the matrix A. Unchanged on exit.

LDA

- INTEGER.
On entry, LDA specifies the first dimension of A as
declared in the calling (sub) program. When
TRANS = 'N' or 'n' then LDA must be at least max(
1, n), otherwise LDA must be at least max(1, k).
Unchanged on exit.

kb is

B

-
COMPLEX*16 array of DIMENSION (LDB, kb), where
k when TRANS = 'N' or 'n', and is n otherwise.
Before entry with TRANS = 'N' or 'n', the leading
n by k part of the array B must contain the matrix
B, otherwise the leading k by n part of the array
B must contain the matrix B. Unchanged on exit.

LDB

- INTEGER.
On entry, LDB specifies the first dimension of B as
declared in the calling (sub) program. When
TRANS = 'N' or 'n' then LDB must be at least max(
1, n), otherwise LDB must be at least max(1, k).

Unchanged on exit.

BETA

- DOUBLE PRECISION.
On entry, BETA specifies the scalar beta. Unchanged
on exit.

C

- COMPLEX*16 array of DIMENSION (LDC, n).
Before entry with UPLO = 'U' or 'u', the leading
n by n upper triangular part of the array C must con-
tain the upper triangular part of the hermitian
matrix and the strictly lower triangular part of C
is not referenced. On exit, the upper triangular
part of the array C is overwritten by the upper tri-
angular part of the updated matrix. Before entry
with UPLO = 'L' or 'l', the leading n by n lower
triangular part of the array C must contain the lower
triangular part of the hermitian matrix and the

strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, n)$. Unchanged on exit.

— zher2k.f —

```

SUBROUTINE ZHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA,
$                C, LDC )
*
* .. Scalar Arguments ..
CHARACTER          TRANS, UPLO
INTEGER            K, LDA, LDB, LDC, N
DOUBLE PRECISION   BETA
COMPLEX*16         ALPHA
*
* ..
* .. Array Arguments ..
COMPLEX*16         A( LDA, * ), B( LDB, * ), C( LDC, * )
*
* ..
*
* Level 3 Blas routine.
*
* -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
* -- Modified 8-Nov-93 to set C(J,J) to DBLE( C(J,J) ) when BETA = 1.
*   Ed Anderson, Cray Research Inc.
*
*
* .. External Functions ..
LOGICAL            LSAME
EXTERNAL           LSAME
*
* ..
* .. External Subroutines ..
EXTERNAL           XERBLA
*
* ..

```

```

*      .. Intrinsic Functions ..
      INTRINSIC          DBLE, DCONJG, MAX
*
*      ..
*      .. Local Scalars ..
      LOGICAL            UPPER
      INTEGER            I, INFO, J, L, NROWA
      COMPLEX*16         TEMP1, TEMP2
*
*      ..
*      .. Parameters ..
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
      COMPLEX*16         ZERO
      PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
        NROWA = N
      ELSE
        NROWA = K
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
      INFO = 0
      IF( ( .NOT.UPPER ) .AND. ( .NOT.LSAME( UPLO, 'L' ) ) ) THEN
        INFO = 1
      ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ) .AND.
$         ( .NOT.LSAME( TRANS, 'C' ) ) ) THEN
        INFO = 2
      ELSE IF( N.LT.0 ) THEN
        INFO = 3
      ELSE IF( K.LT.0 ) THEN
        INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
        INFO = 7
      ELSE IF( LDB.LT.MAX( 1, NROWA ) ) THEN
        INFO = 9
      ELSE IF( LDC.LT.MAX( 1, N ) ) THEN
        INFO = 12
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZHER2K', INFO )
        RETURN
      END IF
*
*      Quick return if possible.
*
      IF( ( N.EQ.0 ) .OR. ( ( ALPHA.EQ.ZERO ) .OR. ( K.EQ.0 ) ) .AND.

```

```

$      ( BETA.EQ.ONE ) ) RETURN
*
*      And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
        IF( UPPER ) THEN
          IF( BETA.EQ.DBLE( ZERO ) ) THEN
            DO 20 J = 1, N
              DO 10 I = 1, J
                C( I, J ) = ZERO
10             CONTINUE
20             CONTINUE
          ELSE
            DO 40 J = 1, N
              DO 30 I = 1, J - 1
                C( I, J ) = BETA*C( I, J )
30             CONTINUE
                C( J, J ) = BETA*DBLE( C( J, J ) )
40             CONTINUE
          END IF
        ELSE
          IF( BETA.EQ.DBLE( ZERO ) ) THEN
            DO 60 J = 1, N
              DO 50 I = J, N
                C( I, J ) = ZERO
50             CONTINUE
60             CONTINUE
          ELSE
            DO 80 J = 1, N
              C( J, J ) = BETA*DBLE( C( J, J ) )
              DO 70 I = J + 1, N
                C( I, J ) = BETA*C( I, J )
70             CONTINUE
80             CONTINUE
          END IF
        END IF
      RETURN
    END IF
*
*      Start the operations.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
*
*      Form C := alpha*A*conjg( B' ) + conjg( alpha )*B*conjg( A' ) +
*      C.
*
      IF( UPPER ) THEN
        DO 130 J = 1, N
          IF( BETA.EQ.DBLE( ZERO ) ) THEN
            DO 90 I = 1, J

```

```

          C( I, J ) = ZERO
90      CONTINUE
      ELSE IF( BETA.NE.ONE ) THEN
          DO 100 I = 1, J - 1
              C( I, J ) = BETA*C( I, J )
100      CONTINUE
          C( J, J ) = BETA*DBLE( C( J, J ) )
      ELSE
          C( J, J ) = DBLE( C( J, J ) )
      END IF
      DO 120 L = 1, K
          IF( ( A( J, L ).NE.ZERO ) .OR. ( B( J, L ).NE.ZERO ) )
              $          THEN
                  TEMP1 = ALPHA*DCONJG( B( J, L ) )
                  TEMP2 = DCONJG( ALPHA*A( J, L ) )
                  DO 110 I = 1, J - 1
                      C( I, J ) = C( I, J ) + A( I, L )*TEMP1 +
110      $          B( I, L )*TEMP2
                  CONTINUE
                  C( J, J ) = DBLE( C( J, J ) ) +
130      $          DBLE( A( J, L )*TEMP1+B( J, L )*TEMP2 )
              END IF
          CONTINUE
120      CONTINUE
130      CONTINUE
      ELSE
          DO 180 J = 1, N
              IF( BETA.EQ.DBLE( ZERO ) ) THEN
                  DO 140 I = J, N
                      C( I, J ) = ZERO
140      CONTINUE
                  ELSE IF( BETA.NE.ONE ) THEN
                      DO 150 I = J + 1, N
                          C( I, J ) = BETA*C( I, J )
150      CONTINUE
                          C( J, J ) = BETA*DBLE( C( J, J ) )
                      ELSE
                          C( J, J ) = DBLE( C( J, J ) )
                      END IF
                      DO 170 L = 1, K
                          IF( ( A( J, L ).NE.ZERO ) .OR. ( B( J, L ).NE.ZERO ) )
                              $          THEN
                                  TEMP1 = ALPHA*DCONJG( B( J, L ) )
                                  TEMP2 = DCONJG( ALPHA*A( J, L ) )
                                  DO 160 I = J + 1, N
                                      C( I, J ) = C( I, J ) + A( I, L )*TEMP1 +
160      $          B( I, L )*TEMP2
                                  CONTINUE
                                      C( J, J ) = DBLE( C( J, J ) ) +
180      $          DBLE( A( J, L )*TEMP1+B( J, L )*TEMP2 )
                              END IF
                          END IF
                      END IF
                  END IF
              END IF
          END IF
      END IF

```

```

170          CONTINUE
180          CONTINUE
          END IF
        ELSE
*
*          Form  $C := \alpha * \text{conjg}(A') * B + \text{conjg}(\alpha) * \text{conjg}(B') * A + C.$ 
*
*          IF( UPPER ) THEN
            DO 210 J = 1, N
              DO 200 I = 1, J
                TEMP1 = ZERO
                TEMP2 = ZERO
                DO 190 L = 1, K
                  TEMP1 = TEMP1 + DCONJG( A( L, I ) ) * B( L, J )
                  TEMP2 = TEMP2 + DCONJG( B( L, I ) ) * A( L, J )
190          CONTINUE
                IF( I.EQ.J ) THEN
                  IF( BETA.EQ.DBLE( ZERO ) ) THEN
                    C( J, J ) = DBLE( ALPHA*TEMP1+DCONJG( ALPHA ) *
$                      TEMP2 )
                    ELSE
                      C( J, J ) = BETA*DBLE( C( J, J ) ) +
$                      DBLE( ALPHA*TEMP1+DCONJG( ALPHA ) *
$                      TEMP2 )
                    END IF
                  ELSE
                    IF( BETA.EQ.DBLE( ZERO ) ) THEN
                      C( I, J ) = ALPHA*TEMP1 + DCONJG( ALPHA ) * TEMP2
                    ELSE
                      C( I, J ) = BETA*C( I, J ) + ALPHA*TEMP1 +
$                      DCONJG( ALPHA ) * TEMP2
                    END IF
                  END IF
                CONTINUE
200          CONTINUE
210          CONTINUE
        ELSE
          DO 240 J = 1, N
            DO 230 I = J, N
              TEMP1 = ZERO
              TEMP2 = ZERO
              DO 220 L = 1, K
                TEMP1 = TEMP1 + DCONJG( A( L, I ) ) * B( L, J )
                TEMP2 = TEMP2 + DCONJG( B( L, I ) ) * A( L, J )
220          CONTINUE
              IF( I.EQ.J ) THEN
                IF( BETA.EQ.DBLE( ZERO ) ) THEN
                  C( J, J ) = DBLE( ALPHA*TEMP1+DCONJG( ALPHA ) *
$                      TEMP2 )
                  ELSE
                    END IF
                CONTINUE
230          CONTINUE
240          CONTINUE
        END IF
      END IF
    CONTINUE
  END IF

```

```

                C( J, J ) = BETA*DBLE( C( J, J ) ) +
$                DBLE( ALPHA*TEMP1+DCONJG( ALPHA )*
$                TEMP2 )
                END IF
            ELSE
                IF( BETA.EQ.DBLE( ZERO ) ) THEN
                    C( I, J ) = ALPHA*TEMP1 + DCONJG( ALPHA )*TEMP2
                ELSE
                    C( I, J ) = BETA*C( I, J ) + ALPHA*TEMP1 +
$                    DCONJG( ALPHA )*TEMP2
                END IF
            END IF
230        CONTINUE
240    CONTINUE
    END IF
END IF
*
    RETURN
*
*    End of ZHER2K.
*
    END

```

— BLAS 3 zher2k —

```

(let* ((one 1.0) (zero (complex 0.0 0.0)))
  (declare (type (double-float 1.0 1.0) one) (type (complex double-float) zero))
  (defun zher2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (double-float) beta)
      (type (simple-array (complex double-float) (*)) c b a)
      (type (complex double-float) alpha)
      (type fixnum ldc ldb$ lda k n)
      (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (b (complex double-float) b-%data% b-%offset%)
       (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0)
        (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
          (type fixnum i info j l nrowa)
          (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)

```

```

      (setf nrowa n))
    (t
      (setf nrowa k)))
  (setf upper (char-equal uplo #\U))
  (setf info 0)
  (cond
    ((and (not upper) (not (char-equal uplo #\L)))
      (setf info 1))
    ((and (not (char-equal trans #\N)) (not (char-equal trans #\C)))
      (setf info 2))
    ((< n 0)
      (setf info 3))
    ((< k 0)
      (setf info 4))
    ((< lda (max (the fixnum 1) (the fixnum nrowa)))
      (setf info 7))
    ((< ldb$
      (max (the fixnum 1) (the fixnum nrowa)))
      (setf info 9))
    ((< ldc (max (the fixnum 1) (the fixnum n)))
      (setf info 12)))
  (cond
    ((/= info 0)
      (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "ZHER2K" info)
      (go end_label)))
  (if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
    (go end_label))
  (cond
    ((= alpha zero)
      (cond
        (upper
          (cond
            ((= beta (coerce (realpart zero) 'double-float))
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
              (tagbody
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i j) nil)
                  (tagbody
                    (setf (f2cl-lib:fref c-%data%
                                          (i j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                      zero))))))
            (t
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
              (tagbody

```



```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j
      (f2cl-lib:int-sub 1)))
    nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%))
    (* beta
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%))))))
(setf (f2cl-lib:fref c-%data%
  (j j)
  ((1 ldc) (1 *))
  c-%offset%))
  (coerce
    (* beta
      (coerce (realpart
        (f2cl-lib:fref c-%data%
          (j j)
          ((1 ldc) (1 *))
          c-%offset%)) 'double-float))
      '(complex double-float))))))
(t
  (cond
    ((= beta (coerce (realpart zero) 'double-float))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
          (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *))
              c-%offset%))
              zero))))))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
            (j j)
            ((1 ldc) (1 *))
            c-%offset%))
            (coerce

```

```

(* beta
  (coerce (realpart
    (f2cl-lib:fref c-%data%
      (j j)
      ((1 ldc) (1 *))
      c-%offset%)) 'double-float))
  '(complex double-float)))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
    (* beta
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%))))))

(go end_label)))
(cond
  ((char-equal trans #\N)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (cond
            ((= beta (coerce (realpart zero) 'double-float))
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i j) nil)
              (tagbody
                (setf (f2cl-lib:fref c-%data%
                  (i j)
                  ((1 ldc) (1 *))
                  c-%offset%)
                  zero))))
            ( /= beta one)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i
                  (f2cl-lib:int-add j
                    (f2cl-lib:int-sub 1)))
                  nil)
                (tagbody
                  (setf (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
                    (* beta

```

```

                                (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%))))
(setf (f2cl-lib:fref c-%data%
                    (j j)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (coerce
        (* beta
          (coerce (realpart
                    (f2cl-lib:fref c-%data%
                                (j j)
                                ((1 ldc) (1 *))
                                c-%offset%)) 'double-float))
          '(complex double-float))))
(t
 (setf (f2cl-lib:fref c-%data%
                     (j j)
                     ((1 ldc) (1 *))
                     c-%offset%)
       (coerce
         (coerce (realpart
                   (f2cl-lib:fref c-%data%
                               (j j)
                               ((1 ldc) (1 *))
                               c-%offset%)) 'double-float)
         '(complex double-float))))
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
              (> 1 k) nil)
(tagbody
 (cond
  ((or (/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
        (/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
   (setf temp1
          (* alpha
             (f2cl-lib:dconjg
              (f2cl-lib:fref b-%data%
                            (j 1)
                            ((1 ldb$) (1 *))
                            b-%offset%))))
      (setf temp2
            (coerce
              (f2cl-lib:dconjg
               (* alpha
                  (f2cl-lib:fref a-%data%
                                (j 1)
                                ((1 lda) (1 *))
                                a-%offset%)))
              '(complex double-float)))

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add j
      (f2cl-lib:int-sub
        1)))
    nil)
(tagbody
  (setf (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
    (+
      (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)
      (*
        (f2cl-lib:fref a-%data%
          (i 1)
          ((1 lda) (1 *))
          a-%offset%)
        temp1)
      (*
        (f2cl-lib:fref b-%data%
          (i 1)
          ((1 ldb$) (1 *))
          b-%offset%)
        temp2))))))
(setf (f2cl-lib:fref c-%data%
  (j j)
  ((1 ldc) (1 *))
  c-%offset%)
  (coerce
    (+
      (coerce (realpart
        (f2cl-lib:fref c-%data%
          (j j)
          ((1 ldc) (1 *))
          c-%offset%)) 'double-float)
      (coerce (realpart
        (+
          (*
            (f2cl-lib:fref a-%data%
              (j 1)
              ((1 lda) (1 *))
              a-%offset%)
            temp1)
          (*
            (f2cl-lib:fref b-%data%
              (j 1)

```

```
((1 ldb$) (1 *))
      b-%offset%)
    temp2))) 'double-float))
  '(complex double-float)))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j n) nil)
(tagbody
(cond
((= beta (coerce (realpart zero) 'double-float))
(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
  ((> i n) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *)))
      c-%offset%)
zero))))
( (/ = beta one)
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  ((> i n) nil)
(tagbody
(setf (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *)))
      c-%offset%)
(* beta
(f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *)))
c-%offset%))))
(setf (f2cl-lib:fref c-%data%
                  (j j)
                  ((1 ldc) (1 *)))
      c-%offset%)
(coerce
(* beta
(coerce (realpart
(f2cl-lib:fref c-%data%
              (j j)
              ((1 ldc) (1 *)))
c-%offset%))' double-float))
'(complex double-float))))
(t
(setf (f2cl-lib:fref c-%data%
                  (j j)
                  ((1 ldc) (1 *)))
      c-%offset%)
(coerce
```

```

(coerce (realpart
  (f2cl-lib:fref c-%data%
    (j j)
    ((1 ldc) (1 *))
    c-%offset%)) 'double-float)
'(complex double-float))))
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  ((> 1 k) nil)
(tagbody
  (cond
    ((or (/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
      (/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
      (setf temp1
        (* alpha
          (f2cl-lib:dconjg
            (f2cl-lib:fref b-%data%
              (j 1)
              ((1 ldb$) (1 *))
              b-%offset%))))
        (setf temp2
          (coerce
            (f2cl-lib:dconjg
              (* alpha
                (f2cl-lib:fref a-%data%
                  (j 1)
                  ((1 lda) (1 *))
                  a-%offset%)))
              ' (complex double-float))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
            (f2cl-lib:int-add i 1))
              ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%))
                (+
                  (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
                  (*
                    (f2cl-lib:fref a-%data%
                      (i 1)
                      ((1 lda) (1 *))
                      a-%offset%)
                    temp1)
                  (*
                    (f2cl-lib:fref b-%data%
                      (i 1)

```

```

                                ((1 ldb$) (1 *))
                                b-%offset%)
                                temp2))))))
(setf (f2cl-lib:fref c-%data%
                    (j j)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (coerce
       (+
        (coerce (realpart
                  (f2cl-lib:fref c-%data%
                                (j j)
                                ((1 ldc) (1 *))
                                c-%offset%)) 'double-float)
        (coerce (realpart
                  (+
                   (*
                    (f2cl-lib:fref a-%data%
                                    (j 1)
                                    ((1 lda) (1 *))
                                    a-%offset%)
                    temp1)
                   (*
                    (f2cl-lib:fref b-%data%
                                    (j 1)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)
                    temp2))) 'double-float))
       ' (complex double-float)))))))))
(t
 (cond
  (upper
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
   (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i j) nil)
    (tagbody
     (setf temp1 zero)
     (setf temp2 zero)
     (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                   ((> l k) nil)
     (tagbody
      (setf temp1
              (+ temp1
                 (*
                  (f2cl-lib:dconjg
                   (f2cl-lib:fref a-%data%
                                   (l i)
                                   ((1 lda) (1 *))

```

```

                                a-%offset%))
(f2cl-lib:fref b-%data%
  (1 j)
  ((1 ldb$) (1 *))
  b-%offset%)))
(setf temp2
  (+ temp2
    (*
      (f2cl-lib:dconjg
        (f2cl-lib:fref b-%data%
          (1 i)
          ((1 ldb$) (1 *))
          b-%offset%))
      (f2cl-lib:fref a-%data%
        (1 j)
        ((1 lda) (1 *))
        a-%offset%))))))
(cond
  ((= i j)
    (cond
      ((= beta (coerce (realpart zero) 'double-float))
        (setf (f2cl-lib:fref c-%data%
          (j j)
          ((1 ldc) (1 *))
          c-%offset%))
          (coerce
            (coerce (realpart
              (+ (* alpha temp1)
                (* (f2cl-lib:dconjg alpha) temp2)))
              'double-float)
            'double-float)))
      (t
        (setf (f2cl-lib:fref c-%data%
          (j j)
          ((1 ldc) (1 *))
          c-%offset%))
          (coerce
            (+
              (* beta
                (coerce (realpart
                  (f2cl-lib:fref c-%data%
                    (j j)
                    ((1 ldc) (1 *))
                    c-%offset%))
                  'double-float))
              (coerce (realpart
                (+ (* alpha temp1)
                  (* (f2cl-lib:dconjg alpha) temp2)))
                'double-float))
            'double-float))))))

```



```

(t
  (cond
    ((= beta (coerce (realpart zero) 'double-float))
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *)))
            c-%offset%))
    (+ (* alpha temp1)
      (* (f2cl-lib:dconjg alpha) temp2))))
(t
  (setf (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *)))
        c-%offset%)
    (+
      (* beta
        (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *)))
        c-%offset%))
      (* alpha temp1)
      (* (f2cl-lib:dconjg alpha) temp2))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf temp1 zero)
          (setf temp2 zero)
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
            ((> l k) nil)
            (tagbody
              (setf temp1
                (+ temp1
                  (*
                     (f2cl-lib:dconjg
                      (f2cl-lib:fref a-%data%
                                      (l i)
                                      ((1 lda) (1 *)))
                                      a-%offset%))
                     (f2cl-lib:fref b-%data%
                                      (l j)
                                      ((1 ldb) (1 *)))
                                      b-%offset%))))
                (setf temp2
                  (+ temp2
                    (*
                     (f2cl-lib:dconjg

```

```
(f2cl-lib:fref b-%data%
              (1 i)
              ((1 ldb$) (1 *))
              b-%offset%))
(f2cl-lib:fref a-%data%
              (1 j)
              ((1 lda) (1 *))
              a-%offset%))))))

(cond
  ((= i j)
    (cond
      ((= beta (coerce (realpart zero) 'double-float))
        (setf (f2cl-lib:fref c-%data%
                          (j j)
                          ((1 ldc) (1 *))
                          c-%offset%))

          (coerce
            (coerce (realpart
                      (+ (* alpha temp1)
                        (* (f2cl-lib:dconjg alpha) temp2)))
              'double-float)
            '(complex double-float))))))
    (t
      (setf (f2cl-lib:fref c-%data%
                        (j j)
                        ((1 ldc) (1 *))
                        c-%offset%)

          (coerce
            (+
              (* beta
                (coerce (realpart
                          (f2cl-lib:fref c-%data%
                                          (j j)
                                          ((1 ldc) (1 *))
                                          c-%offset%))
                  'double-float))
              (coerce (realpart
                        (+ (* alpha temp1)
                          (* (f2cl-lib:dconjg alpha) temp2)))
                          'double-float))
            '(complex double-float))))))
    (t
      (cond
        ((= beta (coerce (realpart zero) 'double-float))
          (setf (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%)

              (+ (* alpha temp1)
                (* (f2cl-lib:dconjg alpha) temp2))))
```

```

(t
  (setf (f2cl-lib:fref c-%data%
                     (i j)
                     ((1 ldc) (1 *)))
        c-%offset%)
  (+
    (* beta
      (f2cl-lib:fref c-%data%
                     (i j)
                     ((1 ldc) (1 *)))
      c-%offset%))
    (* alpha temp1)
    (* (f2cl-lib:dconjg alpha) temp2))))))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil nil))))

```

zherk BLAS

— zherk.input —

```

)set break resume
)sys rm -f zherk.output
)spool zherk.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zherk.help —

```

=====
zherk examples
=====

=====
Man Page Details
=====

NAME

```

ZHERK - perform one of the hermitian rank k operations C
 := alpha*A*conjg(A') + beta*C,

SYNOPSIS

SUBROUTINE ZHERK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
 C, LDC)

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDC

DOUBLE PRECISION ALPHA, BETA

COMPLEX*16 A(LDA, *), C(LDC, *)

PURPOSE

ZHERK performs one of the hermitian rank k operations

or

C := alpha*conjg(A')*A + beta*C,

where alpha and beta are real scalars, C is an n by n
 hermitian matrix and A is an n by k matrix in the first
 case and a k by n matrix in the second case.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or
 lower triangular part of the array C is to be
 referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part
 of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part
 of C is to be referenced.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be per-
 formed as follows:

TRANS = 'N' or 'n' C := alpha*A*conjg(A') +
 beta*C.

TRANS = 'C' or 'c' C := alpha*conjg(A')*A +
 beta*C.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

-
COMPLEX*16 array of DIMENSION (LDA, ka), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

BETA - DOUBLE PRECISION.

On entry, BETA specifies the scalar beta. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n).

Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the

array C is overwritten by the lower triangular part of the updated matrix. Note that the imaginary parts

of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

LDC - INTEGER.

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, n)$. Unchanged on exit.

— zherk.f —

```

SUBROUTINE ZHERK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
*   .. Scalar Arguments ..
CHARACTER          TRANS, UPLO
INTEGER            K, LDA, LDC, N
DOUBLE PRECISION   ALPHA, BETA
*   ..
*   .. Array Arguments ..
COMPLEX*16         A( LDA, * ), C( LDC, * )
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*   -- Modified 8-Nov-93 to set C(J,J) to DBLE( C(J,J) ) when BETA = 1.
*   Ed Anderson, Cray Research Inc.
*
*
*   .. External Functions ..
LOGICAL            LSAME
EXTERNAL           LSAME
*   ..
*   .. External Subroutines ..
EXTERNAL           XERBLA
*   ..
*   .. Intrinsic Functions ..
INTRINSIC          DBLE, DCMPLX, DCONJG, MAX
*   ..

```

```

*      .. Local Scalars ..
LOGICAL          UPPER
INTEGER          I, INFO, J, L, NROWA
DOUBLE PRECISION RTEMP
COMPLEX*16       TEMP
*
*      ..
*      .. Parameters ..
DOUBLE PRECISION ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
          NROWA = N
      ELSE
          NROWA = K
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
      INFO = 0
      IF( ( .NOT.UPPER ) .AND. ( .NOT.LSAME( UPLO, 'L' ) ) ) THEN
          INFO = 1
      ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ) .AND.
$          ( .NOT.LSAME( TRANS, 'C' ) ) ) THEN
          INFO = 2
      ELSE IF( N.LT.0 ) THEN
          INFO = 3
      ELSE IF( K.LT.0 ) THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) ) THEN
          INFO = 7
      ELSE IF( LDC.LT.MAX( 1, N ) ) THEN
          INFO = 10
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZHERK ', INFO )
          RETURN
      END IF
*
*      Quick return if possible.
*
      IF( ( N.EQ.0 ) .OR. ( ( ALPHA.EQ.ZERO ) .OR. ( K.EQ.0 ) ) .AND.
$          ( BETA.EQ.ONE ) ) )RETURN
*
*      And when alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO ) THEN
          IF( UPPER ) THEN

```

```

      IF( BETA.EQ.ZERO ) THEN
        DO 20 J = 1, N
          DO 10 I = 1, J
            C( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
        ELSE
          DO 40 J = 1, N
            DO 30 I = 1, J - 1
              C( I, J ) = BETA*C( I, J )
30            CONTINUE
              C( J, J ) = BETA*DBLE( C( J, J ) )
40            CONTINUE
          END IF
        ELSE
          IF( BETA.EQ.ZERO ) THEN
            DO 60 J = 1, N
              DO 50 I = J, N
                C( I, J ) = ZERO
50              CONTINUE
60              CONTINUE
            ELSE
              DO 80 J = 1, N
                C( J, J ) = BETA*DBLE( C( J, J ) )
                DO 70 I = J + 1, N
                  C( I, J ) = BETA*C( I, J )
70                CONTINUE
80                CONTINUE
              END IF
            END IF
          RETURN
        END IF
*
*      Start the operations.
*
      IF( LSAME( TRANS, 'N' ) ) THEN
*
*      Form C := alpha*A*conjg( A' ) + beta*C.
*
      IF( UPPER ) THEN
        DO 130 J = 1, N
          IF( BETA.EQ.ZERO ) THEN
            DO 90 I = 1, J
              C( I, J ) = ZERO
90            CONTINUE
          ELSE IF( BETA.NE.ONE ) THEN
            DO 100 I = 1, J - 1
              C( I, J ) = BETA*C( I, J )
100            CONTINUE
              C( J, J ) = BETA*DBLE( C( J, J ) )

```



```

ELSE
  C( J, J ) = DBLE( C( J, J ) )
END IF
DO 120 L = 1, K
  IF( A( J, L ).NE.DCMPLX( ZERO ) ) THEN
    TEMP = ALPHA*DCONJG( A( J, L ) )
    DO 110 I = 1, J - 1
      C( I, J ) = C( I, J ) + TEMP*A( I, L )
110    CONTINUE
    C( J, J ) = DBLE( C( J, J ) ) +
      DBLE( TEMP*A( I, L ) )
    $
  END IF
120  CONTINUE
130  CONTINUE
ELSE
  DO 180 J = 1, N
    IF( BETA.EQ.ZERO ) THEN
      DO 140 I = J, N
        C( I, J ) = ZERO
140      CONTINUE
    ELSE IF( BETA.NE.ONE ) THEN
      C( J, J ) = BETA*DBLE( C( J, J ) )
      DO 150 I = J + 1, N
        C( I, J ) = BETA*C( I, J )
150      CONTINUE
    ELSE
      C( J, J ) = DBLE( C( J, J ) )
    END IF
    DO 170 L = 1, K
      IF( A( J, L ).NE.DCMPLX( ZERO ) ) THEN
        TEMP = ALPHA*DCONJG( A( J, L ) )
        C( J, J ) = DBLE( C( J, J ) ) +
          DBLE( TEMP*A( J, L ) )
        $
        DO 160 I = J + 1, N
          C( I, J ) = C( I, J ) + TEMP*A( I, L )
160        CONTINUE
      END IF
    END IF
170    CONTINUE
180    CONTINUE
  END IF
ELSE
  *
  *      Form C := alpha*conjg( A' )*A + beta*C.
  *
  IF( UPPER ) THEN
    DO 220 J = 1, N
      DO 200 I = 1, J - 1
        TEMP = ZERO
        DO 190 L = 1, K
          TEMP = TEMP + DCONJG( A( L, I ) )*A( L, J )

```

```

190          CONTINUE
            IF( BETA.EQ.ZERO ) THEN
              C( I, J ) = ALPHA*TEMP
            ELSE
              C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
            END IF
200          CONTINUE
          RTEMP = ZERO
          DO 210 L = 1, K
            RTEMP = RTEMP + DCONJG( A( L, J ) ) * A( L, J )
210          CONTINUE
          IF( BETA.EQ.ZERO ) THEN
            C( J, J ) = ALPHA*RTEMP
          ELSE
            C( J, J ) = ALPHA*RTEMP + BETA*DBLE( C( J, J ) )
          END IF
220          CONTINUE
        ELSE
          DO 260 J = 1, N
            RTEMP = ZERO
            DO 230 L = 1, K
              RTEMP = RTEMP + DCONJG( A( L, J ) ) * A( L, J )
230            CONTINUE
            IF( BETA.EQ.ZERO ) THEN
              C( J, J ) = ALPHA*RTEMP
            ELSE
              C( J, J ) = ALPHA*RTEMP + BETA*DBLE( C( J, J ) )
            END IF
            DO 250 I = J + 1, N
              TEMP = ZERO
              DO 240 L = 1, K
                TEMP = TEMP + DCONJG( A( L, I ) ) * A( L, J )
240              CONTINUE
              IF( BETA.EQ.ZERO ) THEN
                C( I, J ) = ALPHA*TEMP
              ELSE
                C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
              END IF
            END IF
            CONTINUE
250          CONTINUE
260          CONTINUE
        END IF
      END IF
*
      RETURN
*
*      End of ZHERK .
*
      END

```

— BLAS 3 zherk —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun zherk (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c a)
              (type (double-float) beta alpha)
              (type fixnum ldc lda k n)
              (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (rtemp 0.0) (i 0) (info 0) (j 0) (l 0)
              (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp)
                  (type (double-float) rtemp)
                  (type fixnum i info j l nrowa)
                  (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
           (setf nrowa n))
          (t
           (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
           (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\C)))
           (setf info 2))
          ((< n 0)
           (setf info 3))
          ((< k 0)
           (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))
           (setf info 7))
          ((< ldc (max (the fixnum 1) (the fixnum n)))
           (setf info 10)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "ZHERK" info)
            (go end_label)))

```

```

(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
  (go end_label))
(cond
  ((= alpha zero)
   (cond
     (upper
      (cond
        ((= beta zero)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
           ((> j n) nil)
          (tagbody
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i j) nil)
             (tagbody
               (setf (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%)
                 (coerce zero '(complex double-float)))))))
          (t
           (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
             ((> j n) nil)
            (tagbody
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i
                  (f2cl-lib:int-add j
                    (f2cl-lib:int-sub 1)))
                 nil)
               (tagbody
                 (setf (f2cl-lib:fref c-%data%
                                       (i j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)
                   (* beta
                    (f2cl-lib:fref c-%data%
                                      (i j)
                                      ((1 ldc) (1 *))
                                      c-%offset%))))))
              (setf (f2cl-lib:fref c-%data%
                                  (j j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                (coerce
                  (* beta
                   (coerce (realpart
                           (f2cl-lib:fref c-%data%
                                           (j j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)) 'double-float))
                  '(complex double-float))))))

```

```

(t
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%))
                (coerce zero '(complex double-float))))))))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                                (j j)
                                ((1 ldc) (1 *))
                                c-%offset%))
              (coerce
                (* beta
                  (coerce (realpart
                           (f2cl-lib:fref c-%data%
                                           (j j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)) 'double-float))
                  '(complex double-float)))
              (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                            (f2cl-lib:int-add i 1))
                ((> i n) nil)
                (tagbody
                  (setf (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%))
                      (* beta
                        (f2cl-lib:fref c-%data%
                                        (i j)
                                        ((1 ldc) (1 *))
                                        c-%offset%))))))))
      (go end_label)))
  )
(cond
  ((char-equal trans #\N)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j n) nil)

```

```

(tagbody
  (cond
    ((= beta zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i j) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (coerce zero '(complex double-float))))))
    ((/= beta one)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub 1)))
          nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *))
                               c-%offset%)
                (* beta
                  (f2cl-lib:fref c-%data%
                                   (i j)
                                   ((1 ldc) (1 *))
                                   c-%offset%))))))
      (setf (f2cl-lib:fref c-%data%
                          (j j)
                          ((1 ldc) (1 *))
                          c-%offset%)
            (coerce
              (* beta
                (coerce (realpart
                          (f2cl-lib:fref c-%data%
                                           (j j)
                                           ((1 ldc) (1 *))
                                           c-%offset%)) 'double-float))
              '(complex double-float))))))
    (t
      (setf (f2cl-lib:fref c-%data%
                          (j j)
                          ((1 ldc) (1 *))
                          c-%offset%)
            (coerce
              (coerce (realpart
                        (f2cl-lib:fref c-%data%
                                       (j j)
                                       ((1 ldc) (1 *))
                                       c-%offset%)) 'double-float)
              '(complex double-float))))))

```

```

      '(complex double-float))))))
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
(tagbody
  (cond
    ((/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *)))
      (coerce (complex zero) '(complex double-float)))
      (setf temp
        (coerce
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (j 1)
                ((1 lda) (1 *))
                a-%offset%)))
          '(complex double-float)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i
          (f2cl-lib:int-add j
            (f2cl-lib:int-sub
              1)))
          nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)
          (+
            (f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *))
              c-%offset%)
            (* temp
              (f2cl-lib:fref a-%data%
                (i 1)
                ((1 lda) (1 *))
                a-%offset%))))))
      (setf (f2cl-lib:fref c-%data%
        (j j)
        ((1 ldc) (1 *))
        c-%offset%)
        (coerce
          (+
            (coerce (realpart
              (f2cl-lib:fref c-%data%
                (j j)
                ((1 ldc) (1 *))
                c-%offset%)) 'double-float)
            (coerce (realpart
              (* temp

```

```

                                (f2c1-lib:fref a-%data%
                                  (i 1)
                                  ((1 lda) (1 *))
                                  a-%offset%)))
                                'double-float))
                                '(complex double-float)))))))))
(t
  (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (cond
        ((= beta zero)
          (f2c1-lib:fdo (i j (f2c1-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf (f2c1-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (coerce zero '(complex double-float))))))
          (coerce zero '(complex double-float))))))
        ((/= beta one)
          (setf (f2c1-lib:fref c-%data%
            (j j)
            ((1 ldc) (1 *))
            c-%offset%)
            (coerce
              (* beta
                (coerce (realpart
                  (f2c1-lib:fref c-%data%
                    (j j)
                    ((1 ldc) (1 *))
                    c-%offset%) 'double-float))
                '(complex double-float))))
            (f2c1-lib:fdo (i (f2c1-lib:int-add j 1)
              (f2c1-lib:int-add i 1))
                ((> i n) nil)
                (tagbody
                  (setf (f2c1-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
                    (* beta
                      (f2c1-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
            (t
              (setf (f2c1-lib:fref c-%data%
                (j j)
                ((1 ldc) (1 *))

```



```

                                c-%offset%)
      (coerce
        (coerce (realpart
          (f2cl-lib:fref c-%data%
            (j j)
            ((1 ldc) (1 *))
            c-%offset%)) 'double-float)
          '(complex double-float))))
    (f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
      (> 1 k) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *)))
        (coerce (complex zero) 'complex double-float)))
      (setf temp
        (coerce
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (j 1)
                ((1 lda) (1 *))
                a-%offset%)))
            'complex double-float)))
      (setf (f2cl-lib:fref c-%data%
        (j j)
        ((1 ldc) (1 *))
        c-%offset%)
        (coerce
          (+
            (coerce (realpart
              (f2cl-lib:fref c-%data%
                (j j)
                ((1 ldc) (1 *))
                c-%offset%)) 'double-float)
            (coerce (realpart
              (* temp
                (f2cl-lib:fref a-%data%
                  (j 1)
                  ((1 lda) (1 *))
                  a-%offset%)))
                'double-float))
            'complex double-float)))
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add i 1))
        (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)

```

```

(+
  (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* temp
    (f2cl-lib:fref a-%data%
      (i 1)
      ((1 lda) (1 *))
      a-%offset%)))))))))
(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i
              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              nil)
            (tagbody
              (setf temp (coerce zero '(complex double-float)))
              (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
                ((> l k) nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2cl-lib:dconjg
                          (f2cl-lib:fref a-%data%
                            (l i)
                            ((1 lda) (1 *))
                            a-%offset%))
                        (f2cl-lib:fref a-%data%
                          (l j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
                  (cond
                    ((= beta zero)
                     (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
                       (* alpha temp)))
                    (t
                     (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
                       (+ (* alpha temp)

```

```

(* beta
  (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%))))))
(setf rtemp zero)
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
              ((> 1 k) nil)
  (tagbody
    (setf rtemp
      (coerce
        (realpart
          (+ rtemp
            (*
              (f2cl-lib:dconjg
                (f2cl-lib:fref a-%data%
                              (1 j)
                              ((1 lda) (1 *))
                              a-%offset%))
              (f2cl-lib:fref a-%data%
                              (1 j)
                              ((1 lda) (1 *))
                              a-%offset%))))
          'double-float))))
    (cond
      ((= beta zero)
        (setf (f2cl-lib:fref c-%data%
                              (j j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              (coerce (* alpha rtemp) 'complex double-float))))
      (t
        (setf (f2cl-lib:fref c-%data%
                              (j j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              (coerce
                (+ (* alpha rtemp)
                  (* beta
                    (coerce (realpart
                              (f2cl-lib:fref c-%data%
                                              (j j)
                                              ((1 ldc) (1 *))
                                              c-%offset%)
                              'double-float)))
                  'complex double-float))))))
        (t
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        ((> j n) nil)
            (tagbody

```

```

(setf rtemp zero)
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
(tagbody
  (setf rtemp
    (coerce
      (realpart
        (+ rtemp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (1 j)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fref a-%data%
              (1 j)
              ((1 lda) (1 *))
              a-%offset%))))
        'double-float))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (j j)
      ((1 ldc) (1 *))
      c-%offset%)
      (coerce (* alpha rtemp) '(complex double-float))))
(t
  (setf (f2cl-lib:fref c-%data%
    (j j)
    ((1 ldc) (1 *))
    c-%offset%)
    (coerce
      (+ (* alpha rtemp)
        (* beta
          (coerce (realpart
            (f2cl-lib:fref c-%data%
              (j j)
              ((1 ldc) (1 *))
              c-%offset%)
            'double-float)))
        '(complex double-float))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
  (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf temp (coerce zero '(complex double-float)))
  (f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
    (> 1 k) nil)
  (tagbody
    (setf temp

```

```

(+ temp
  (*
    (f2cl-lib:dconjg
      (f2cl-lib:fref a-%data%
        (1 i)
        ((1 lda) (1 *))
        a-%offset%))
    (f2cl-lib:fref a-%data%
      (1 j)
      ((1 lda) (1 *))
      a-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (* alpha temp)))
  (t
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+ (* alpha temp)
        (* beta
          (f2cl-lib:fref c-%data%
            (i j)
            ((1 ldc) (1 *))
            c-%offset%))))))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil)))))

```

zsymm BLAS

— zsymm.input —

```

)set break resume
)sys rm -f zsymm.output
)spool zsymm.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— zsymm.help —

=====

zsymm examples

=====

=====

Man Page Details

=====

NAME

ZSYMM - perform one of the matrix-matrix operations $C := \alpha * A * B + \beta * C$,

SYNOPSIS

SUBROUTINE ZSYMM (SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
BETA, C, LDC)

CHARACTER*1 SIDE, UPLO

INTEGER M, N, LDA, LDB, LDC

COMPLEX*16 ALPHA, BETA

COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE

ZSYMM performs one of the matrix-matrix operations

or

$C := \alpha * B * A + \beta * C$,

where alpha and beta are scalars, A is a symmetric matrix
and B and C are m by n matrices.

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether the symmetric
matrix A appears on the left or right in the
operation as follows:

SIDE = 'L' or 'l' $C := \alpha * A * B + \beta * C$,

SIDE = 'R' or 'r' $C := \alpha * B * A + \beta * C$,

Unchanged on exit.

UPLO - CHARACTER*1.
On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.

Unchanged on exit.

M - INTEGER.
On entry, M specifies the number of rows of the matrix C. M must be at least zero. Unchanged on exit.

N - INTEGER.
On entry, N specifies the number of columns of the matrix C. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .
On entry, ALPHA specifies the scalar alpha.
Unchanged on exit.

ka is

A -
COMPLEX*16 array of DIMENSION (LDA, ka), where m when SIDE = 'L' or 'l' and is n otherwise. Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the sym-

metric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. Unchanged on exit.

- LDA - INTEGER.
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least $\max(1, m)$, otherwise LDA must be at least $\max(1, n)$. Unchanged on exit.
- B - COMPLEX*16 array of DIMENSION (LDB, n).
Before entry, the leading m by n part of the array B must contain the matrix B. Unchanged on exit.
- LDB - INTEGER.
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least $\max(1, m)$. Unchanged on exit.
- BETA - COMPLEX*16 .
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input. Unchanged on exit.
- C - COMPLEX*16 array of DIMENSION (LDC, n).
Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.
- LDC - INTEGER.
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least $\max(1, m)$. Unchanged on exit.

— zsymm.f —

```

SUBROUTINE ZSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB,
$                BETA, C, LDC )
*    .. Scalar Arguments ..

```



```

      CHARACTER*1      SIDE, UPLO
      INTEGER          M, N, LDA, LDB, LDC
      COMPLEX*16       ALPHA, BETA
*    .. Array Arguments ..
      COMPLEX*16       A( LDA, * ), B( LDB, * ), C( LDC, * )
*    ..
*
*    Level 3 Blas routine.
*
*    -- Written on 8-February-1989.
*       Jack Dongarra, Argonne National Laboratory.
*       Iain Duff, AERE Harwell.
*       Jeremy Du Croz, Numerical Algorithms Group Ltd.
*       Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*    .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*    .. External Subroutines ..
      EXTERNAL         XERBLA
*    .. Intrinsic Functions ..
      INTRINSIC        MAX
*    .. Local Scalars ..
      LOGICAL          UPPER
      INTEGER          I, INFO, J, K, NROWA
      COMPLEX*16       TEMP1, TEMP2
*    .. Parameters ..
      COMPLEX*16       ONE
      PARAMETER        ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16       ZERO
      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*    ..
*    .. Executable Statements ..
*
*    Set NROWA as the number of rows of A.
*
      IF( LSAME( SIDE, 'L' ) )THEN
        NROWA = M
      ELSE
        NROWA = N
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
*    Test the input parameters.
*
      INFO = 0
      IF(      ( .NOT.LSAME( SIDE, 'L' ) ).AND.
$          ( .NOT.LSAME( SIDE, 'R' ) ) )THEN
        INFO = 1

```

```

      ELSE IF( ( .NOT.UPPER
$          ( .NOT.LSAME( UPLO, 'L' ) ) ) )THEN
          INFO = 2
      ELSE IF( M .LT.0 )THEN
          INFO = 3
      ELSE IF( N .LT.0 )THEN
          INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
          INFO = 7
      ELSE IF( LDB.LT.MAX( 1, M ) )THEN
          INFO = 9
      ELSE IF( LDC.LT.MAX( 1, M ) )THEN
          INFO = 12
      END IF
      IF( INFO.NE.0 )THEN
          CALL XERBLA( 'ZSYMM ', INFO )
          RETURN
      END IF
*
*   Quick return if possible.
*
      IF( ( M.EQ.0 ).OR.( N.EQ.0 ).OR.
$      ( ( ALPHA.EQ.ZERO ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*   And when alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
          IF( BETA.EQ.ZERO )THEN
              DO 20, J = 1, N
                  DO 10, I = 1, M
                      C( I, J ) = ZERO
10              CONTINUE
20              CONTINUE
          ELSE
              DO 40, J = 1, N
                  DO 30, I = 1, M
                      C( I, J ) = BETA*C( I, J )
30              CONTINUE
40              CONTINUE
          END IF
          RETURN
      END IF
*
*   Start the operations.
*
      IF( LSAME( SIDE, 'L' ) )THEN
*
*       Form C := alpha*A*B + beta*C.
*

```

```

      IF( UPPER )THEN
        DO 70, J = 1, N
          DO 60, I = 1, M
            TEMP1 = ALPHA*B( I, J )
            TEMP2 = ZERO
            DO 50, K = 1, I - 1
              C( K, J ) = C( K, J ) + TEMP1      *A( K, I )
              TEMP2      = TEMP2      + B( K, J )*A( K, I )
50          CONTINUE
            IF( BETA.EQ.ZERO )THEN
              C( I, J ) = TEMP1*A( I, I ) + ALPHA*TEMP2
            ELSE
              C( I, J ) = BETA *C( I, J ) +
$              TEMP1*A( I, I ) + ALPHA*TEMP2
            END IF
60          CONTINUE
70        CONTINUE
      ELSE
        DO 100, J = 1, N
          DO 90, I = M, 1, -1
            TEMP1 = ALPHA*B( I, J )
            TEMP2 = ZERO
            DO 80, K = I + 1, M
              C( K, J ) = C( K, J ) + TEMP1      *A( K, I )
              TEMP2      = TEMP2      + B( K, J )*A( K, I )
80          CONTINUE
            IF( BETA.EQ.ZERO )THEN
              C( I, J ) = TEMP1*A( I, I ) + ALPHA*TEMP2
            ELSE
              C( I, J ) = BETA *C( I, J ) +
$              TEMP1*A( I, I ) + ALPHA*TEMP2
            END IF
90          CONTINUE
100         CONTINUE
        END IF
      ELSE
*
*       Form C := alpha*B*A + beta*C.
*
        DO 170, J = 1, N
          TEMP1 = ALPHA*A( J, J )
          IF( BETA.EQ.ZERO )THEN
            DO 110, I = 1, M
              C( I, J ) = TEMP1*B( I, J )
110          CONTINUE
          ELSE
            DO 120, I = 1, M
              C( I, J ) = BETA*C( I, J ) + TEMP1*B( I, J )
120          CONTINUE
        END IF

```

```

DO 140, K = 1, J - 1
  IF( UPPER )THEN
    TEMP1 = ALPHA*A( K, J )
  ELSE
    TEMP1 = ALPHA*A( J, K )
  END IF
  DO 130, I = 1, M
    C( I, J ) = C( I, J ) + TEMP1*B( I, K )
130  CONTINUE
140  CONTINUE
DO 160, K = J + 1, N
  IF( UPPER )THEN
    TEMP1 = ALPHA*A( J, K )
  ELSE
    TEMP1 = ALPHA*A( K, J )
  END IF
  DO 150, I = 1, M
    C( I, J ) = C( I, J ) + TEMP1*B( I, K )
150  CONTINUE
160  CONTINUE
170  CONTINUE
END IF
*
RETURN
*
* End of ZSYMM .
*
END

```

— BLAS 3 zsymm —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
(declare (type (complex double-float) one) (type (complex double-float) zero))
(defun zsymm (side uplo m n alpha a lda b ldb$ beta c ldc)
  (declare (type (simple-array (complex double-float) (*)) c b a)
    (type (complex double-float) beta alpha)
    (type fixnum ldc ldb$ lda n m)
    (type character uplo side))
  (f2cl-lib:with-multi-array-data
    ((side character side-%data% side-%offset%)
     (uplo character uplo-%data% uplo-%offset%)
     (a (complex double-float) a-%data% a-%offset%)
     (b (complex double-float) b-%data% b-%offset%)
     (c (complex double-float) c-%data% c-%offset%))
    (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0)
          (nrowa 0) (upper nil))

```

```

(declare (type (complex double-float) temp1 temp2)
  (type fixnum i info j k nrowa)
  (type (member t nil) upper))

(cond
  ((char-equal side #\L)
   (setf nrowa m))
  (t
   (setf nrowa n)))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
  ((and (not (char-equal side #\L)) (not (char-equal side #\R)))
   (setf info 1))
  ((and (not upper) (not (char-equal uplo #\L)))
   (setf info 2))
  ((< m 0)
   (setf info 3))
  ((< n 0)
   (setf info 4))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
   (setf info 7))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
   (setf info 9))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
   (setf info 12)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZSYMM" info)
   (go end_label)))
(if (or (= m 0) (= n 0) (and (= alpha zero) (= beta one)))
  (go end_label))
(cond
  ((= alpha zero)
   (cond
    ((= beta zero)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
      (tagbody
       (setf (f2cl-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *))
                           c-%offset%)
              zero))))))
    (t
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

                                (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
      (* beta
        (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))
  (go end_label)))
(cond
  ((char-equal side #\L)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i m) nil)
            (tagbody
              (setf temp1
                (* alpha
                  (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)))
                (setf temp2 zero)
                (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  (> k
                    (f2cl-lib:int-add i
                      (f2cl-lib:int-sub
                        1)))
                    nil)
                  (tagbody
                    (setf (f2cl-lib:fref c-%data%
                                          (k j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                      (+
                        (f2cl-lib:fref c-%data%
                                          (k j)
                                          ((1 ldc) (1 *))
                                          c-%offset%)
                        (* temp1
                          (f2cl-lib:fref a-%data%

```

```

                                (k i)
                                ((1 lda) (1 *))
                                a-%offset%)))
    (setf temp2
      (+ temp2
        (*
          (f2cl-lib:fref b-%data%
                        (k j)
                        ((1 ldb$) (1 *))
                        b-%offset%)
          (f2cl-lib:fref a-%data%
                        (k i)
                        ((1 lda) (1 *))
                        a-%offset%))))))
    (cond
      ((= beta zero)
        (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
          (+
            (* temp1
              (f2cl-lib:fref a-%data%
                            (i i)
                            ((1 lda) (1 *))
                            a-%offset%))
            (* alpha temp2))))))
      (t
        (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
          (+
            (* beta
              (f2cl-lib:fref c-%data%
                            (i j)
                            ((1 ldc) (1 *))
                            c-%offset%))
            (* temp1
              (f2cl-lib:fref a-%data%
                            (i i)
                            ((1 lda) (1 *))
                            a-%offset%))
            (* alpha temp2)))))))))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
            ((> i 1) nil)

```

```

(tagbody
  (setf temp1
    (* alpha
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)))
  (setf temp2 zero)
  (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
    (f2cl-lib:int-add k 1))
    ((> k m) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
        (k j)
        ((1 ldc) (1 *))
        c-%offset%)
        (+
          (f2cl-lib:fref c-%data%
            (k j)
            ((1 ldc) (1 *))
            c-%offset%)
          (* temp1
            (f2cl-lib:fref a-%data%
              (k i)
              ((1 lda) (1 *))
              a-%offset%))))
      (setf temp2
        (+ temp2
          (*
            (f2cl-lib:fref b-%data%
              (k j)
              ((1 ldb$) (1 *))
              b-%offset%)
            (f2cl-lib:fref a-%data%
              (k i)
              ((1 lda) (1 *))
              a-%offset%))))))
  (cond
    ((= beta zero)
      (setf (f2cl-lib:fref c-%data%
        (i j)
        ((1 ldc) (1 *))
        c-%offset%)
        (+
          (* temp1
            (f2cl-lib:fref a-%data%
              (i i)
              ((1 lda) (1 *))
              a-%offset%))
          (* alpha temp2))))

```



```

(t
  (setf (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *)))
        c-%offset%)
    (+
      (* beta
        (f2cl-lib:fref c-%data%
                      (i j)
                      ((1 ldc) (1 *)))
        c-%offset%)
      (* temp1
        (f2cl-lib:fref a-%data%
                      (i i)
                      ((1 lda) (1 *)))
        a-%offset%)
      (* alpha temp2)))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
                        (j j)
                        ((1 lda) (1 *)))
          a-%offset%)))
      (cond
        ((= beta zero)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *)))
                    c-%offset%)
                (* temp1
                  (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *)))
                  b-%offset%))))))
      (t
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *)))
                  c-%offset%)
                (+

```

```

(* beta
  (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%))

(* temp1
  (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))
                b-%offset%)))))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  ((> k (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
   nil)
(tagbody
  (cond
    (upper
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
                        (k j)
                        ((1 lda) (1 *))
                        a-%offset%))))

      (t
        (setf temp1
          (* alpha
            (f2cl-lib:fref a-%data%
                          (j k)
                          ((1 lda) (1 *))
                          a-%offset%))))))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
          (+
            (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)

            (* temp1
              (f2cl-lib:fref b-%data%
                            (i k)
                            ((1 ldb$) (1 *))
                            b-%offset%))))))
        (f2cl-lib:fdo (k (f2cl-lib:int-add j 1) (f2cl-lib:int-add k 1))
          ((> k n) nil)
          (tagbody
            (cond

```

```

      (upper
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
                        (j k)
                        ((1 lda) (1 *))
                        a-%offset%))))
      (t
      (setf temp1
        (* alpha
          (f2cl-lib:fref a-%data%
                        (k j)
                        ((1 lda) (1 *))
                        a-%offset%))))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i m) nil)
      (tagbody
      (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (+
          (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* temp1
            (f2cl-lib:fref b-%data%
                          (i k)
                          ((1 ldb) (1 *))
                          b-%offset%))))))
      end_label
      (return (values nil nil nil nil nil nil nil nil nil nil nil))))

```

zsyr2k BLAS

— zsyr2k.input —

```

)set break resume
)sys rm -f zsyr2k.output
)spool zsyr2k.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— zsy2k.help —

```
=====
zsy2k examples
=====
```

```
=====
Man Page Details
=====
```

NAME

ZSYR2K - perform one of the symmetric rank 2k operations $C := \alpha A^* B' + \alpha B^* A' + \beta C$,

SYNOPSIS

```
SUBROUTINE ZSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB,
                  BETA, C, LDC )
```

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDB, LDC

COMPLEX*16 ALPHA, BETA

COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE

ZSYR2K performs one of the symmetric rank 2k operations

or

$C := \alpha A^* B + \alpha B^* A + \beta C$,

where α and β are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $C := \alpha * A * B' + \alpha * B * A' + \beta * C$.

TRANS = 'T' or 't' $C := \alpha * A' * B + \alpha * B' * A + \beta * C$.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrices A and B. K must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha.

Unchanged on exit.

ka is

A -

COMPLEX*16 array of DIMENSION (LDA, ka), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k). Unchanged on exit.

`kb` is
`B` -
 COMPLEX*16 array of DIMENSION (`LDB`, `kb`), where
 `k` when `TRANS` = 'N' or 'n', and is `n` otherwise.
 Before entry with `TRANS` = 'N' or 'n', the leading
 `n` by `k` part of the array `B` must contain the matrix
 `B`, otherwise the leading `k` by `n` part of the array
 `B` must contain the matrix `B`. Unchanged on exit.

`LDB` - INTEGER.
 On entry, `LDB` specifies the first dimension of `B` as
 declared in the calling (sub) program. When
 `TRANS` = 'N' or 'n' then `LDB` must be at least `max(`
 `1, n)`, otherwise `LDB` must be at least `max(1, k)`.
 Unchanged on exit.

`BETA` - COMPLEX*16 .
 On entry, `BETA` specifies the scalar `beta`. Unchanged
 on exit.

`C` - COMPLEX*16 array of DIMENSION (`LDC`, `n`).
 Before entry with `UPLO` = 'U' or 'u', the leading
 `n` by `n` upper triangular part of the array `C` must con-
 tain the upper triangular part of the symmetric
 matrix and the strictly lower triangular part of `C`
 is not referenced. On exit, the upper triangular
 part of the array `C` is overwritten by the upper tri-
 angular part of the updated matrix. Before entry
 with `UPLO` = 'L' or 'l', the leading `n` by `n` lower
 triangular part of the array `C` must contain the lower
 triangular part of the symmetric matrix and the
 strictly upper triangular part of `C` is not refer-
 enced. On exit, the lower triangular part of the
 array `C` is overwritten by the lower triangular part
 of the updated matrix.

`LDC` - INTEGER.
 On entry, `LDC` specifies the first dimension of `C` as
 declared in the calling (sub) program. `LDC`
 must be at least `max(1, n)`. Unchanged on exit.

— zsyr2k.f —

SUBROUTINE ZSYR2K(`UPLO`, `TRANS`, `N`, `K`, `ALPHA`, `A`, `LDA`, `B`, `LDB`,
 \$ `BETA`, `C`, `LDC`)

```

*      .. Scalar Arguments ..
      CHARACTER*1      UPLO, TRANS
      INTEGER           N, K, LDA, LDB, LDC
      COMPLEX*16        ALPHA, BETA
*      .. Array Arguments ..
      COMPLEX*16        A( LDA, * ), B( LDB, * ), C( LDC, * )
*
*
*      Level 3 Blas routine.
*
*      -- Written on 8-February-1989.
*      Jack Dongarra, Argonne National Laboratory.
*      Iain Duff, AERE Harwell.
*      Jeremy Du Croz, Numerical Algorithms Group Ltd.
*      Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
*      .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL           LSAME
*      .. External Subroutines ..
      EXTERNAL           XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC          MAX
*      .. Local Scalars ..
      LOGICAL            UPPER
      INTEGER            I, INFO, J, L, NROWA
      COMPLEX*16         TEMP1, TEMP2
*      .. Parameters ..
      COMPLEX*16         ONE
      PARAMETER          ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16         ZERO
      PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      IF( LSAME( TRANS, 'N' ) )THEN
        NROWA = N
      ELSE
        NROWA = K
      END IF
      UPPER = LSAME( UPLO, 'U' )
*
      INFO = 0
      IF( ( .NOT.UPPER ) .AND.
$        ( .NOT.LSAME( UPLO, 'L' ) ) )THEN
        INFO = 1
      ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ) .AND.

```

```

$      ( .NOT.LSAME( TRANS, 'T' ) ) )THEN
      INFO = 2
      ELSE IF( N .LT.0 )THEN
        INFO = 3
      ELSE IF( K .LT.0 )THEN
        INFO = 4
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
        INFO = 7
      ELSE IF( LDB.LT.MAX( 1, NROWA ) )THEN
        INFO = 9
      ELSE IF( LDC.LT.MAX( 1, N ) )THEN
        INFO = 12
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZSYR2K', INFO )
        RETURN
      END IF
*
*      Quick return if possible.
*
      IF( ( N.EQ.0 ).OR.
$      ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.0 ) ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*      And when alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
        IF( UPPER )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 20, J = 1, N
              DO 10, I = 1, J
                C( I, J ) = ZERO
10              CONTINUE
20              CONTINUE
          ELSE
            DO 40, J = 1, N
              DO 30, I = 1, J
                C( I, J ) = BETA*C( I, J )
30              CONTINUE
40              CONTINUE
          END IF
        ELSE
          IF( BETA.EQ.ZERO )THEN
            DO 60, J = 1, N
              DO 50, I = J, N
                C( I, J ) = ZERO
50              CONTINUE
60              CONTINUE
          ELSE
            DO 80, J = 1, N

```



```

DO 70, I = J, N
    C( I, J ) = BETA*C( I, J )
70    CONTINUE
80    CONTINUE
    END IF
    END IF
    RETURN
END IF

*
*   Start the operations.
*
IF( LSAME( TRANS, 'N' ) )THEN
*
*   Form C := alpha*A*B' + alpha*B*A' + C.
*
    IF( UPPER )THEN
        DO 130, J = 1, N
            IF( BETA.EQ.ZERO )THEN
                DO 90, I = 1, J
                    C( I, J ) = ZERO
90                CONTINUE
            ELSE IF( BETA.NE.ONE )THEN
                DO 100, I = 1, J
                    C( I, J ) = BETA*C( I, J )
100                CONTINUE
            END IF
            DO 120, L = 1, K
                IF( ( A( J, L ).NE.ZERO ).OR.
$                   ( B( J, L ).NE.ZERO ) )THEN
                    TEMP1 = ALPHA*B( J, L )
                    TEMP2 = ALPHA*A( J, L )
                    DO 110, I = 1, J
                        C( I, J ) = C( I, J ) + A( I, L )*TEMP1 +
$                                           B( I, L )*TEMP2
110                CONTINUE
            END IF
120            CONTINUE
130        CONTINUE
    ELSE
        DO 180, J = 1, N
            IF( BETA.EQ.ZERO )THEN
                DO 140, I = J, N
                    C( I, J ) = ZERO
140                CONTINUE
            ELSE IF( BETA.NE.ONE )THEN
                DO 150, I = J, N
                    C( I, J ) = BETA*C( I, J )
150                CONTINUE
            END IF
            DO 170, L = 1, K

```

```

      IF( ( A( J, L ).NE.ZERO ).OR.
$      ( B( J, L ).NE.ZERO ) )THEN
          TEMP1 = ALPHA*B( J, L )
          TEMP2 = ALPHA*A( J, L )
          DO 160, I = J, N
              C( I, J ) = C( I, J ) + A( I, L )*TEMP1 +
$              B( I, L )*TEMP2
160          CONTINUE
          END IF
170      CONTINUE
180      CONTINUE
      END IF
      ELSE
*
*      Form C := alpha*A'*B + alpha*B'*A + C.
*
      IF( UPPER )THEN
          DO 210, J = 1, N
              DO 200, I = 1, J
                  TEMP1 = ZERO
                  TEMP2 = ZERO
                  DO 190, L = 1, K
                      TEMP1 = TEMP1 + A( L, I )*B( L, J )
                      TEMP2 = TEMP2 + B( L, I )*A( L, J )
190                  CONTINUE
                  IF( BETA.EQ.ZERO )THEN
                      C( I, J ) = ALPHA*TEMP1 + ALPHA*TEMP2
                  ELSE
                      C( I, J ) = BETA *C( I, J ) +
$                      ALPHA*TEMP1 + ALPHA*TEMP2
                  END IF
200                  CONTINUE
210              CONTINUE
          ELSE
              DO 240, J = 1, N
                  DO 230, I = J, N
                      TEMP1 = ZERO
                      TEMP2 = ZERO
                      DO 220, L = 1, K
                          TEMP1 = TEMP1 + A( L, I )*B( L, J )
                          TEMP2 = TEMP2 + B( L, I )*A( L, J )
220                      CONTINUE
                      IF( BETA.EQ.ZERO )THEN
                          C( I, J ) = ALPHA*TEMP1 + ALPHA*TEMP2
                      ELSE
                          C( I, J ) = BETA *C( I, J ) +
$                          ALPHA*TEMP1 + ALPHA*TEMP2
                      END IF
230                      CONTINUE
240                  CONTINUE

```

```

        END IF
    END IF
*
    RETURN
*
*   End of ZSYR2K.
*
END

```

— BLAS 3 zsyr2k —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zsyr2k (uplo trans n k alpha a lda b ldb$ beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c b a)
      (type (complex double-float) beta alpha)
      (type fixnum ldc ldb$ lda k n)
      (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (b (complex double-float) b-%data% b-%offset%)
       (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp1 #C(0.0 0.0)) (temp2 #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0)
        (nrowa 0) (upper nil))
        (declare (type (complex double-float) temp1 temp2)
          (type fixnum i info j l nrowa)
          (type (member t nil) upper))
        (cond
          ((char-equal trans #\N)
            (setf nrowa n))
          (t
            (setf nrowa k)))
        (setf upper (char-equal uplo #\U))
        (setf info 0)
        (cond
          ((and (not upper) (not (char-equal uplo #\L)))
            (setf info 1))
          ((and (not (char-equal trans #\N)) (not (char-equal trans #\T)))
            (setf info 2))
          ((< n 0)
            (setf info 3))
          ((< k 0)
            (setf info 4))
          ((< lda (max (the fixnum 1) (the fixnum nrowa)))

```

```

(setf info 7))
(<< ldb$
  (max (the fixnum 1) (the fixnum nrowa)))
(setf info 9))
(<< ldc (max (the fixnum 1) (the fixnum n)))
(setf info 12)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "ZSYR2K" info)
  (go end_label)))
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
  (go end_label))
(cond
  (= alpha zero)
  (cond
    (upper
      (cond
        (= beta zero)
        (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
            (> i j) nil)
          (tagbody
            (setf (f2c1-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *)))
              c-%offset%
              zero))))))
    (t
      (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
          (> i j) nil)
          (tagbody
            (setf (f2c1-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *)))
              c-%offset%
              (* beta
                (f2c1-lib:fref c-%data%
                               (i j)
                               ((1 ldc) (1 *)))
              c-%offset%))))))))
    (t
      (cond
        (= beta zero)

```

```

(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
        zero))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (* beta
          (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))))))))))
  (go end_label)))
(cond
  ((char-equal trans #\N)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (cond
            ((= beta zero)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i j) nil)
              (tagbody
                (setf (f2cl-lib:fref c-%data%
                                    (i j)
                                    ((1 ldc) (1 *))
                                    c-%offset%)
                  zero))))
            ((/= beta one)
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i j) nil)
              (tagbody
                (setf (f2cl-lib:fref c-%data%

```

```

        (i j)
        ((1 ldc) (1 *))
        c-%offset%)

(* beta
  (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%))))))
(f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1 k) nil)
(tagbody
  (cond
    ((or (/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
      (/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
      (setf temp1
        (* alpha
          (f2cl-lib:fref b-%data%
            (j 1)
            ((1 ldb$) (1 *))
            b-%offset%)))

      (setf temp2
        (* alpha
          (f2cl-lib:fref a-%data%
            (j 1)
            ((1 lda) (1 *))
            a-%offset%)))

      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i j) nil)

      (tagbody
        (setf (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)

          (+
            (f2cl-lib:fref c-%data%
              (i j)
              ((1 ldc) (1 *))
              c-%offset%)

            (*
              (f2cl-lib:fref a-%data%
                (i 1)
                ((1 lda) (1 *))
                a-%offset%)

              temp1)

            (*
              (f2cl-lib:fref b-%data%
                (i 1)
                ((1 ldb$) (1 *))
                b-%offset%)

              temp2))))))))))

```

```

(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((= beta zero)
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
          (> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                zero))))
      ((/= beta one)
        (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
          (> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                (* beta
                  (f2cl-lib:fref c-%data%
                                (i j)
                                ((1 ldc) (1 *))
                                c-%offset%))))))
    (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
      (> l k) nil)
    (tagbody
      (cond
        ((or (/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
              (/= (f2cl-lib:fref b (j 1) ((1 ldb$) (1 *))) zero))
          (setf temp1
            (* alpha
              (f2cl-lib:fref b-%data%
                              (j 1)
                              ((1 ldb$) (1 *))
                              b-%offset%)))
          (setf temp2
            (* alpha
              (f2cl-lib:fref a-%data%
                              (j 1)
                              ((1 lda) (1 *))
                              a-%offset%)))
          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
            (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                  (i j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                  (+ temp1 temp2))))
        (t
          (tagbody
            (setf (f2cl-lib:fref c-%data%
                                  (i j)
                                  ((1 ldc) (1 *))
                                  c-%offset%)
                  (+ temp1 temp2))))))
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                              (i j)
                              ((1 ldc) (1 *))
                              c-%offset%)
              (+ temp1 temp2))))))

```

```

((1 ldc) (1 *))
c-%offset%)
(+
  (f2cl-lib:fref c-%data%
    (i j)
    ((1 ldc) (1 *))
    c-%offset%)
  (*
    (f2cl-lib:fref a-%data%
      (i 1)
      ((1 lda) (1 *))
      a-%offset%)
    temp1)
  (*
    (f2cl-lib:fref b-%data%
      (i 1)
      ((1 ldb$) (1 *))
      b-%offset%)
    temp2)))))))))))))
(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i j) nil)
        (tagbody
          (setf temp1 zero)
          (setf temp2 zero)
          (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
            ((> l k) nil)
          (tagbody
            (setf temp1
              (+ temp1
                (*
                  (f2cl-lib:fref a-%data%
                    (l i)
                    ((1 lda) (1 *))
                    a-%offset%)
                  (f2cl-lib:fref b-%data%
                    (l j)
                    ((1 ldb$) (1 *))
                    b-%offset%))))))
            (setf temp2
              (+ temp2
                (*
                  (f2cl-lib:fref b-%data%
                    (l i)
                    ((1 ldb$) (1 *))

```



```

                                b-%offset%)
(f2cl-lib:fref a-%data%
  (1 j)
  ((1 lda) (1 *))
  a-%offset%))))))
(cond
  ((= beta zero)
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+ (* alpha temp1) (* alpha temp2))))
  (t
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (+
        (* beta
          (f2cl-lib:fref c-%data%
            (i j)
            ((1 ldc) (1 *))
            c-%offset%))
        (* alpha temp1)
        (* alpha temp2))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf temp1 zero)
      (setf temp2 zero)
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
        (> l k) nil)
        (tagbody
          (setf temp1
            (+ temp1
              (*
                (f2cl-lib:fref a-%data%
                  (1 i)
                  ((1 lda) (1 *))
                  a-%offset%)
                (f2cl-lib:fref b-%data%
                  (1 j)
                  ((1 ldb) (1 *))
                  b-%offset%))))
            (setf temp2
              (+ temp2

```

```

(*
  (f2cl-lib:fref b-%data%
                (1 i)
                ((1 ldb$) (1 *)))
  b-%offset%)
(f2cl-lib:fref a-%data%
                (1 j)
                ((1 lda) (1 *)))
a-%offset%))))))
(cond
  ((= beta zero)
   (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *)))
         c-%offset%)
   (+ (* alpha temp1) (* alpha temp2))))
(t
  (setf (f2cl-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *)))
        c-%offset%)
  (+
   (* beta
    (f2cl-lib:fref c-%data%
                  (i j)
                  ((1 ldc) (1 *)))
    c-%offset%)
   (* alpha temp1)
   (* alpha temp2)))))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil nil)))))

```

zsyrk BLAS

— zsyrk.input —

```

)set break resume
)sys rm -f zsyrk.output
)spool zsyrk.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— zsyрк.help —

=====

zsyрк examples

=====

=====

Man Page Details

=====

NAME

ZSYRK - perform one of the symmetric rank k operations C
:= alpha*A*A' + beta*C,

SYNOPSIS

SUBROUTINE ZSYRK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA,
 C, LDC)

CHARACTER*1 UPLO, TRANS

INTEGER N, K, LDA, LDC

COMPLEX*16 ALPHA, BETA

COMPLEX*16 A(LDA, *), C(LDC, *)

PURPOSE

ZSYRK performs one of the symmetric rank k operations

or

C := alpha*A'*A + beta*C,

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

PARAMETERS

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS - CHARACTER*1.

On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n' $C := \alpha * A * A' + \beta * C$.

TRANS = 'T' or 't' $C := \alpha * A' * A + \beta * C$.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the order of the matrix C. N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

ka is

A

- COMPLEX*16 array of DIMENSION (LDA, ka), where k when TRANS = 'N' or 'n', and is n otherwise. Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least $\max(1, n)$, otherwise LDA must be at least $\max(1, k)$. Unchanged on exit.

BETA - COMPLEX*16 .

On entry, BETA specifies the scalar beta. Unchanged on exit.

C - COMPLEX*16 array of DIMENSION (LDC, n).
 Before entry with UPLO = 'U' or 'u', the leading
 n by n upper triangular part of the array C must con-
 tain the upper triangular part of the symmetric
 matrix and the strictly lower triangular part of C
 is not referenced. On exit, the upper triangular
 part of the array C is overwritten by the upper tri-
 angular part of the updated matrix. Before entry
 with UPLO = 'L' or 'l', the leading n by n lower
 triangular part of the array C must contain the lower
 triangular part of the symmetric matrix and the
 strictly upper triangular part of C is not refer-
 enced. On exit, the lower triangular part of the
 array C is overwritten by the lower triangular part
 of the updated matrix.

LDC - INTEGER.

On entry, LDC specifies the first dimension of C as
 declared in the calling (sub) program. LDC
 must be at least max(1, n). Unchanged on exit.

— zsyrk.f —

```

SUBROUTINE ZSYRK ( UPLO, TRANS, N, K, ALPHA, A, LDA,
$                BETA, C, LDC )
*   .. Scalar Arguments ..
CHARACTER*1      UPLO, TRANS
INTEGER          N, K, LDA, LDC
COMPLEX*16       ALPHA, BETA
*   .. Array Arguments ..
COMPLEX*16       A( LDA, * ), C( LDC, * )
*
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*   .. External Functions ..
LOGICAL          LSAME

```

```

EXTERNAL          LSAME
* .. External Subroutines ..
EXTERNAL          XERBLA
* .. Intrinsic Functions ..
INTRINSIC          MAX
* .. Local Scalars ..
LOGICAL            UPPER
INTEGER            I, INFO, J, L, NROWA
COMPLEX*16         TEMP
* .. Parameters ..
COMPLEX*16         ONE
PARAMETER          ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16         ZERO
PARAMETER          ( ZERO = ( 0.0D+0, 0.0D+0 ) )
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
IF( LSAME( TRANS, 'N' ) )THEN
    NROWA = N
ELSE
    NROWA = K
END IF
UPPER = LSAME( UPLO, 'U' )
*
INFO = 0
IF(      ( .NOT.UPPER                ) .AND.
$      ( .NOT.LSAME( UPLO, 'L' ) )      )THEN
    INFO = 1
ELSE IF( ( .NOT.LSAME( TRANS, 'N' ) ) .AND.
$      ( .NOT.LSAME( TRANS, 'T' ) )      )THEN
    INFO = 2
ELSE IF( N .LT.0                )THEN
    INFO = 3
ELSE IF( K .LT.0                )THEN
    INFO = 4
ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
    INFO = 7
ELSE IF( LDC.LT.MAX( 1, N      ) )THEN
    INFO = 10
END IF
IF( INFO.NE.0 )THEN
    CALL XERBLA( 'ZSYRK ', INFO )
    RETURN
END IF
*
* Quick return if possible.
*
IF( ( N.EQ.0 ) .OR.

```

```

$      ( ( ( ALPHA.EQ.ZERO ).OR.( K.EQ.O ) ).AND.( BETA.EQ.ONE ) ) )
$      RETURN
*
*      And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
        IF( UPPER )THEN
          IF( BETA.EQ.ZERO )THEN
            DO 20, J = 1, N
              DO 10, I = 1, J
                C( I, J ) = ZERO
10             CONTINUE
20             CONTINUE
          ELSE
            DO 40, J = 1, N
              DO 30, I = 1, J
                C( I, J ) = BETA*C( I, J )
30             CONTINUE
40             CONTINUE
          END IF
        ELSE
          IF( BETA.EQ.ZERO )THEN
            DO 60, J = 1, N
              DO 50, I = J, N
                C( I, J ) = ZERO
50             CONTINUE
60             CONTINUE
          ELSE
            DO 80, J = 1, N
              DO 70, I = J, N
                C( I, J ) = BETA*C( I, J )
70             CONTINUE
80             CONTINUE
          END IF
        END IF
        RETURN
      END IF
*
*      Start the operations.
*
      IF( LSAME( TRANS, 'N' ) )THEN
*
*      Form  C := alpha*A*A' + beta*C.
*
        IF( UPPER )THEN
          DO 130, J = 1, N
            IF( BETA.EQ.ZERO )THEN
              DO 90, I = 1, J
                C( I, J ) = ZERO
90             CONTINUE

```

```

ELSE IF( BETA.NE.ONE )THEN
  DO 100, I = 1, J
    C( I, J ) = BETA*C( I, J )
100    CONTINUE
  END IF
  DO 120, L = 1, K
    IF( A( J, L ).NE.ZERO )THEN
      TEMP = ALPHA*A( J, L )
      DO 110, I = 1, J
        C( I, J ) = C( I, J ) + TEMP*A( I, L )
110      CONTINUE
      END IF
    CONTINUE
120  CONTINUE
130  CONTINUE
ELSE
  DO 180, J = 1, N
    IF( BETA.EQ.ZERO )THEN
      DO 140, I = J, N
        C( I, J ) = ZERO
140      CONTINUE
      ELSE IF( BETA.NE.ONE )THEN
        DO 150, I = J, N
          C( I, J ) = BETA*C( I, J )
150        CONTINUE
        END IF
        DO 170, L = 1, K
          IF( A( J, L ).NE.ZERO )THEN
            TEMP = ALPHA*A( J, L )
            DO 160, I = J, N
              C( I, J ) = C( I, J ) + TEMP*A( I, L )
160            CONTINUE
            END IF
          CONTINUE
170        CONTINUE
180      CONTINUE
    END IF
  ELSE
*
*    Form C := alpha*A'*A + beta*C.
*
    IF( UPPER )THEN
      DO 210, J = 1, N
        DO 200, I = 1, J
          TEMP = ZERO
          DO 190, L = 1, K
            TEMP = TEMP + A( L, I )*A( L, J )
190          CONTINUE
          IF( BETA.EQ.ZERO )THEN
            C( I, J ) = ALPHA*TEMP
          ELSE
            C( I, J ) = ALPHA*TEMP + BETA*C( I, J )

```



```

                END IF
200             CONTINUE
210         CONTINUE
        ELSE
            DO 240, J = 1, N
                DO 230, I = J, N
                    TEMP = ZERO
                    DO 220, L = 1, K
                        TEMP = TEMP + A( L, I ) * A( L, J )
220                 CONTINUE
                    IF( BETA.EQ.ZERO ) THEN
                        C( I, J ) = ALPHA*TEMP
                    ELSE
                        C( I, J ) = ALPHA*TEMP + BETA*C( I, J )
                    END IF
                END IF
230             CONTINUE
240         CONTINUE
        END IF
    END IF
*
    RETURN
*
*   End of ZSYRK .
*
    END

```

— BLAS 3 zsyrc —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun zsyrc (uplo trans n k alpha a lda beta c ldc)
    (declare (type (simple-array (complex double-float) (*)) c a)
              (type (complex double-float) beta alpha)
              (type fixnum ldc lda k n)
              (type character trans uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (c (complex double-float) c-%data% c-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (l 0) (nrowa 0)
             (upper nil))
        (declare (type (complex double-float) temp)
                  (type fixnum i info j l nrowa)
                  (type (member t nil) upper))
        (cond

```

```

((char-equal trans #\N)
 (setf nrowa n))
(t
 (setf nrowa k)))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
 ((and (not upper) (not (char-equal uplo #\L)))
  (setf info 1))
 ((and (not (char-equal trans #\N)) (not (char-equal trans #\T)))
  (setf info 2))
 (< n 0)
 (setf info 3))
 (< k 0)
 (setf info 4))
 (< lda (max (the fixnum 1) (the fixnum nrowa)))
 (setf info 7))
 (< ldc (max (the fixnum 1) (the fixnum n)))
 (setf info 10)))
(cond
 (/= info 0)
 (error
  " ** On entry to ~a parameter number ~a had an illegal value~%"
  "ZSYRK" info)
 (go end_label)))
(if (or (= n 0) (and (or (= alpha zero) (= k 0)) (= beta one)))
 (go end_label))
(cond
 (= alpha zero)
 (cond
 (upper
  (cond
  (= beta zero)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
   (> j n) nil)
  (tagbody
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i j) nil)
   (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%)
      zero))))))
  (t
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
   (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     (> i j) nil)

```

```

        (tagbody
          (setf (f2cl-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%))
                (* beta
                  (f2cl-lib:fref c-%data%
                                 (i j)
                                 ((1 ldc) (1 *))
                                 c-%offset%)))))))))
(t
 (cond
  ((= beta zero)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
   (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                  ((> i n) nil)
    (tagbody
     (setf (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *))
                          c-%offset%))
            zero))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
 (tagbody
  (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                ((> i n) nil)
  (tagbody
   (setf (f2cl-lib:fref c-%data%
                        (i j)
                        ((1 ldc) (1 *))
                        c-%offset%))
          (* beta
            (f2cl-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *))
                           c-%offset%)))))))))
  (go end_label)))
(cond
 ((char-equal trans #\N)
  (cond
   (upper
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  ((> j n) nil)
    (tagbody
     (cond
      ((= beta zero)

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i j) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      zero))))
(/= beta one)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i j) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (* beta
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%))))))
(f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
  (> l k) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref a (j 1) ((1 lda) (1 *))) zero)
        (setf temp
          (* alpha
            (f2cl-lib:fref a-%data%
              (j 1)
              ((1 lda) (1 *))
              a-%offset%)))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i j) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%
                (i j)
                ((1 ldc) (1 *))
                c-%offset%)
                (+
                  (f2cl-lib:fref c-%data%
                    (i j)
                    ((1 ldc) (1 *))
                    c-%offset%)
                  (* temp
                    (f2cl-lib:fref a-%data%
                      (i 1)
                      ((1 lda) (1 *))
                      a-%offset%))))))))))

```

(t

[illegible]

```

(f2c1-lib:fref a-%data%
  (i 1)
  ((1 lda) (1 *))
  a-%offset%)))))))))))))
(t
  (cond
    (upper
      (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
            ((> i j) nil)
            (tagbody
              (setf temp zero)
              (f2c1-lib:fdo (l 1 (f2c1-lib:int-add l 1))
                ((> l k) nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2c1-lib:fref a-%data%
                          (l i)
                          ((1 lda) (1 *))
                          a-%offset%)
                        (f2c1-lib:fref a-%data%
                          (l j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
                  (cond
                    ((= beta zero)
                     (setf (f2c1-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
                       (* alpha temp)))
                    (t
                     (setf (f2c1-lib:fref c-%data%
                       (i j)
                       ((1 ldc) (1 *))
                       c-%offset%)
                       (+ (* alpha temp)
                         (* beta
                           (f2c1-lib:fref c-%data%
                             (i j)
                             ((1 ldc) (1 *))
                             c-%offset%))))))
                  (t
                    (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
                      ((> j n) nil)
                      (tagbody

```

```

(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf temp zero)
  (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
    (> l k) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:fref a-%data%
              (l i)
              ((1 lda) (1 *))
              a-%offset%)
            (f2cl-lib:fref a-%data%
              (l j)
              ((1 lda) (1 *))
              a-%offset%))))))
      (cond
        ((= beta zero)
          (setf (f2cl-lib:fref c-%data%
            (i j)
            ((1 ldc) (1 *))
            c-%offset%)
            (* alpha temp)))
        (t
          (setf (f2cl-lib:fref c-%data%
            (i j)
            ((1 ldc) (1 *))
            c-%offset%)
            (+ (* alpha temp)
              (* beta
                (f2cl-lib:fref c-%data%
                  (i j)
                  ((1 ldc) (1 *))
                  c-%offset%))))))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil))

```

ztrmm BLAS

— ztrmm.input —

)set break resume

```

)sys rm -f ztrmm.output
)spool ztrmm.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztrmm.help —

```

=====
ztrmm examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ZTRMM - perform one of the matrix-matrix operations $B := \alpha * \text{op}(A) * B$, or $B := \alpha * B * \text{op}(A)$ where α is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A)$ is one of $\text{op}(A) = A$ or $\text{op}(A) = A'$ or $\text{op}(A) = \text{conjg}(A')$

SYNOPSIS

```

SUBROUTINE ZTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
                  LDA, B, LDB )

```

```

CHARACTER*1 SIDE, UPLO, TRANSA, DIAG

```

```

INTEGER      M, N, LDA, LDB

```

```

COMPLEX*16   ALPHA

```

```

COMPLEX*16   A( LDA, * ), B( LDB, * )

```

PURPOSE

ZTRMM performs one of the matrix-matrix operations

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether $\text{op}(A)$ multiplies B from the left or right as follows:

SIDE = 'L' or 'l' $B := \alpha * \text{op}(A) * B$.

SIDE = 'R' or 'r' $B := \alpha * B * \text{op}(A)$.

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' $\text{op}(A) = A$.

TRANSA = 'T' or 't' $\text{op}(A) = A'$.

TRANSA = 'C' or 'c' $\text{op}(A) = \text{conjg}(A')$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16.

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need

not be set before entry. Unchanged on exit.

is m

A

-
COMPLEX*16 array of DIMENSION (LDA, k), where k
when SIDE = 'L' or 'l' and is n when SIDE = 'R'
or 'r'. Before entry with UPLO = 'U' or 'u', the
leading k by k upper triangular part of the array A
must contain the upper triangular matrix and the
strictly lower triangular part of A is not refer-
enced. Before entry with UPLO = 'L' or 'l', the
leading k by k lower triangular part of the array A
must contain the lower triangular matrix and the
strictly upper triangular part of A is not refer-
enced. Note that when DIAG = 'U' or 'u', the diag-
onal elements of A are not referenced either, but
are assumed to be unity. Unchanged on exit.

LDA

- INTEGER.
On entry, LDA specifies the first dimension of A as
declared in the calling (sub) program. When SIDE =
'L' or 'l' then LDA must be at least max(1, m),
when SIDE = 'R' or 'r' then LDA must be at least
max(1, n). Unchanged on exit.

B

- COMPLEX*16 array of DIMENSION (LDB, n).
Before entry, the leading m by n part of the array
B must contain the matrix B, and on exit is
overwritten by the transformed matrix.

LDB

- INTEGER.
On entry, LDB specifies the first dimension of B as
declared in the calling (sub) program. LDB
must be at least max(1, m). Unchanged on exit.

— ztrmm.f —

```

SUBROUTINE ZTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
$               B, LDB )
*   .. Scalar Arguments ..
CHARACTER*1     SIDE, UPLO, TRANSA, DIAG
INTEGER         M, N, LDA, LDB
COMPLEX*16      ALPHA
*   .. Array Arguments ..
COMPLEX*16      A( LDA, * ), B( LDB, * )
*   ..

```

```

*
* Level 3 Blas routine.
*
* -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*
* .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
* .. External Subroutines ..
EXTERNAL         XERBLA
* .. Intrinsic Functions ..
INTRINSIC        DCONJG, MAX
* .. Local Scalars ..
LOGICAL          LSIDE, NOCONJ, NOUNIT, UPPER
INTEGER          I, INFO, J, K, NROWA
COMPLEX*16       TEMP
* .. Parameters ..
COMPLEX*16       ONE
PARAMETER        ( ONE = ( 1.0D+0, 0.0D+0 ) )
COMPLEX*16       ZERO
PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
LSIDE = LSAME( SIDE , 'L' )
IF( LSIDE )THEN
    NROWA = M
ELSE
    NROWA = N
END IF
NOCONJ = LSAME( TRANSA, 'T' )
NOUNIT = LSAME( DIAG , 'N' )
UPPER = LSAME( UPLO , 'U' )
*
INFO = 0
IF( ( .NOT.LSIDE ) )AND.
$ ( .NOT.LSAME( SIDE , 'R' ) ) )THEN
    INFO = 1
ELSE IF( ( .NOT.UPPER ) )AND.
$ ( .NOT.LSAME( UPLO , 'L' ) ) )THEN
    INFO = 2
ELSE IF( ( .NOT.LSAME( TRANSA, 'N' ) ) )AND.
$ ( .NOT.LSAME( TRANSA, 'T' ) ) )AND.

```

```

$          ( .NOT.LSAME( TRANSA, 'C' ) ) )THEN
      INFO = 3
      ELSE IF( ( .NOT.LSAME( DIAG , 'U' ) ).AND.
$          ( .NOT.LSAME( DIAG , 'N' ) ) )THEN
      INFO = 4
      ELSE IF( M .LT.0 )THEN
      INFO = 5
      ELSE IF( N .LT.0 )THEN
      INFO = 6
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN
      INFO = 9
      ELSE IF( LDB.LT.MAX( 1, M ) )THEN
      INFO = 11
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZTRMM ', INFO )
        RETURN
      END IF
*
*      Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
*      And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
        DO 20, J = 1, N
          DO 10, I = 1, M
            B( I, J ) = ZERO
10      CONTINUE
20      CONTINUE
        RETURN
      END IF
*
*      Start the operations.
*
      IF( LSIDE )THEN
        IF( LSAME( TRANSA, 'N' ) )THEN
*
*          Form  B := alpha*A*B.
*
          IF( UPPER )THEN
            DO 50, J = 1, N
              DO 40, K = 1, M
                IF( B( K, J ).NE.ZERO )THEN
                  TEMP = ALPHA*B( K, J )
                  DO 30, I = 1, K - 1
                    B( I, J ) = B( I, J ) + TEMP*A( I, K )
30      CONTINUE

```

```

                                IF( NOUNIT )
$                                TEMP = TEMP*A( K, K )
                                B( K, J ) = TEMP
                                END IF
40                                CONTINUE
50                                CONTINUE
                                ELSE
                                DO 80, J = 1, N
                                DO 70 K = M, 1, -1
                                IF( B( K, J ).NE.ZERO )THEN
                                TEMP      = ALPHA*B( K, J )
                                B( K, J ) = TEMP
                                IF( NOUNIT )
$                                B( K, J ) = B( K, J )*A( K, K )
                                DO 60, I = K + 1, M
                                B( I, J ) = B( I, J ) + TEMP*A( I, K )
60                                CONTINUE
                                END IF
70                                CONTINUE
80                                CONTINUE
                                END IF
                                ELSE
*
*                                Form B := alpha*A'*B   or   B := alpha*conjg( A' )*B.
*
                                IF( UPPER )THEN
                                DO 120, J = 1, N
                                DO 110, I = M, 1, -1
                                TEMP = B( I, J )
                                IF( NOCONJ )THEN
                                IF( NOUNIT )
$                                TEMP = TEMP*A( I, I )
                                DO 90, K = 1, I - 1
                                TEMP = TEMP + A( K, I )*B( K, J )
90                                CONTINUE
                                ELSE
                                IF( NOUNIT )
$                                TEMP = TEMP*DCONJG( A( I, I ) )
                                DO 100, K = 1, I - 1
                                TEMP = TEMP + DCONJG( A( K, I ) )*B( K, J )
100                               CONTINUE
                                END IF
                                B( I, J ) = ALPHA*TEMP
110                               CONTINUE
120                               CONTINUE
                                ELSE
                                DO 160, J = 1, N
                                DO 150, I = 1, M
                                TEMP = B( I, J )
                                IF( NOCONJ )THEN

```

```

      IF( NOUNIT )
$          TEMP = TEMP*A( I, I )
          DO 130, K = I + 1, M
              TEMP = TEMP + A( K, I )*B( K, J )
130          CONTINUE
          ELSE
              IF( NOUNIT )
$                  TEMP = TEMP*DCONJG( A( I, I ) )
                  DO 140, K = I + 1, M
                      TEMP = TEMP + DCONJG( A( K, I ) )*B( K, J )
140                  CONTINUE
                  END IF
                  B( I, J ) = ALPHA*TEMP
150              CONTINUE
160          CONTINUE
          END IF
      END IF
  ELSE
      IF( LSAME( TRANSA, 'N' ) )THEN
*
*          Form B := alpha*B*A.
*
      IF( UPPER )THEN
          DO 200, J = N, 1, -1
              TEMP = ALPHA
              IF( NOUNIT )
$                  TEMP = TEMP*A( J, J )
              DO 170, I = 1, M
                  B( I, J ) = TEMP*B( I, J )
170              CONTINUE
              DO 190, K = 1, J - 1
                  IF( A( K, J ).NE.ZERO )THEN
                      TEMP = ALPHA*A( K, J )
                      DO 180, I = 1, M
                          B( I, J ) = B( I, J ) + TEMP*B( I, K )
180                      CONTINUE
                      END IF
                  CONTINUE
              DO 200, I = 1, M
                  B( I, J ) = TEMP*B( I, J )
200              CONTINUE
          ELSE
              DO 240, J = 1, N
                  TEMP = ALPHA
                  IF( NOUNIT )
$                      TEMP = TEMP*A( J, J )
                  DO 210, I = 1, M
                      B( I, J ) = TEMP*B( I, J )
210                  CONTINUE
                  DO 230, K = J + 1, N
                      IF( A( K, J ).NE.ZERO )THEN
                          TEMP = ALPHA*A( K, J )

```

```

DO 220, I = 1, M
    B( I, J ) = B( I, J ) + TEMP*B( I, K )
220    CONTINUE
        END IF
230    CONTINUE
240    CONTINUE
        END IF
    ELSE
*
*       Form B := alpha*B*A'   or   B := alpha*B*conjg( A' ).
*
        IF( UPPER )THEN
            DO 280, K = 1, N
                DO 260, J = 1, K - 1
                    IF( A( J, K ).NE.ZERO )THEN
                        IF( NOCONJ )THEN
                            TEMP = ALPHA*A( J, K )
                        ELSE
                            TEMP = ALPHA*DCONJG( A( J, K ) )
                        END IF
                        DO 250, I = 1, M
                            B( I, J ) = B( I, J ) + TEMP*B( I, K )
250                        CONTINUE
                    END IF
260                CONTINUE
                TEMP = ALPHA
                IF( NUNIT )THEN
                    IF( NOCONJ )THEN
                        TEMP = TEMP*A( K, K )
                    ELSE
                        TEMP = TEMP*DCONJG( A( K, K ) )
                    END IF
                END IF
                IF( TEMP.NE.ONE )THEN
                    DO 270, I = 1, M
                        B( I, K ) = TEMP*B( I, K )
270                    CONTINUE
                END IF
280            CONTINUE
        ELSE
            DO 320, K = N, 1, -1
                DO 300, J = K + 1, N
                    IF( A( J, K ).NE.ZERO )THEN
                        IF( NOCONJ )THEN
                            TEMP = ALPHA*A( J, K )
                        ELSE
                            TEMP = ALPHA*DCONJG( A( J, K ) )
                        END IF
                        DO 290, I = 1, M
                            B( I, J ) = B( I, J ) + TEMP*B( I, K )

```

```

290          CONTINUE
          END IF
300      CONTINUE
      TEMP = ALPHA
      IF( NOUNIT )THEN
          IF( NOCONJ )THEN
              TEMP = TEMP*A( K, K )
          ELSE
              TEMP = TEMP*DCONJG( A( K, K ) )
          END IF
      END IF
      IF( TEMP.NE.ONE )THEN
          DO 310, I = 1, M
              B( I, K ) = TEMP*B( I, K )
310      CONTINUE
          END IF
320      CONTINUE
      END IF
  END IF
END IF
*
  RETURN
*
* End of ZTRMM .
*
  END

```

— BLAS 3 ztrmm —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun ztrmm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array (complex double-float) (*)) b a)
      (type (complex double-float) alpha)
      (type fixnum ldb$ lda n m)
      (type character diag transa uplo side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (uplo character uplo-%data% uplo-%offset%)
       (transa character transa-%data% transa-%offset%)
       (diag character diag-%data% diag-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (b (complex double-float) b-%data% b-%offset%))
      (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
             (lside nil) (noconj nil) (nounit nil) (upper nil))
        (declare (type (complex double-float) temp)

```



```

        (type fixnum i info j k nrowa)
        (type (member t nil) lside noconj nunit upper))
(setf lside (char-equal side #\L))
(cond
  (lside
    (setf nrowa m))
  (t
    (setf nrowa n)))
(setf noconj (char-equal transa #\T))
(setf nunit (char-equal diag #\N))
(setf upper (char-equal uplo #\U))
(setf info 0)
(cond
  ((and (not lside) (not (char-equal side #\R)))
    (setf info 1))
  ((and (not upper) (not (char-equal uplo #\L)))
    (setf info 2))
  ((and (not (char-equal transa #\N))
        (not (char-equal transa #\T))
        (not (char-equal transa #\C)))
    (setf info 3))
  ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
    (setf info 4))
  ((< m 0)
    (setf info 5))
  ((< n 0)
    (setf info 6))
  ((< lda (max (the fixnum 1) (the fixnum nrowa)))
    (setf info 9))
  ((< ldb$ (max (the fixnum 1) (the fixnum m)))
    (setf info 11)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "ZTRMM" info)
    (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= alpha zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%

```

```

                                zero))))))
    (go end_label)))
  (cond
    (lside
      (cond
        ((char-equal transa #\N)
          (cond
            (upper
              (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j n) nil)
              (tagbody
                (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
                  ((> k m) nil)
                  (tagbody
                    (cond
                      ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
                        (setf temp
                          (* alpha
                            (f2cl-lib:fref b-%data%
                              (k j)
                              ((1 ldb$) (1 *))
                              b-%offset%)))
                        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i
                            (f2cl-lib:int-add k
                              (f2cl-lib:int-sub
                                1)))
                            nil)
                          (tagbody
                            (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
                              (+
                                (f2cl-lib:fref b-%data%
                                  (i j)
                                  ((1 ldb$) (1 *))
                                  b-%offset%)
                                (* temp
                                  (f2cl-lib:fref a-%data%
                                    (i k)
                                    ((1 lda) (1 *))
                                    a-%offset%))))))
                          (if nunit
                            (setf temp
                              (* temp
                                (f2cl-lib:fref a-%data%
                                  (k k)
                                  ((1 lda) (1 *))
                                  a-%offset%))))))

```

```

        (setf (f2cl-lib:fref b-%data%
                           (k j)
                           ((1 ldb$) (1 *)))
              b-%offset%)
        temp)))))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
               ((> j n) nil)
  (tagbody
   (f2cl-lib:fdo (k m
                  (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
                 ((> k 1) nil)
    (tagbody
     (cond
      ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
       (setf temp
              (* alpha
                 (f2cl-lib:fref b-%data%
                                (k j)
                                ((1 ldb$) (1 *)))
                 b-%offset%)))
       (setf (f2cl-lib:fref b-%data%
                           (k j)
                           ((1 ldb$) (1 *)))
              b-%offset%)
       temp)
      (if nunit
         (setf (f2cl-lib:fref b-%data%
                               (k j)
                               ((1 ldb$) (1 *)))
                b-%offset%)
         (*
          (f2cl-lib:fref b-%data%
                        (k j)
                        ((1 ldb$) (1 *)))
          b-%offset%
          (f2cl-lib:fref a-%data%
                        (k k)
                        ((1 lda) (1 *)))
          a-%offset%))))))
    (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                    (f2cl-lib:int-add i 1))
                  ((> i m) nil)
     (tagbody
      (setf (f2cl-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *)))
            b-%offset%)
      (+
       (f2cl-lib:fref b-%data%

```

```

                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                                (* temp
                                (f2c1-lib:fref a-%data%
                                (i k)
                                ((1 lda) (1 *))
                                a-%offset%))))))))))))))
(t
(cond
  (upper
    (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
      ((> j n) nil)
    (tagbody
      (f2c1-lib:fdo (i m
        (f2c1-lib:int-add i (f2c1-lib:int-sub 1)))
        ((> i 1) nil)
      (tagbody
        (setf temp
          (f2c1-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%))
        (cond
          (noconj
            (if nounit
              (setf temp
                (* temp
                  (f2c1-lib:fref a-%data%
                    (i i)
                    ((1 lda) (1 *))
                    a-%offset%))))
              (f2c1-lib:fdo (k 1 (f2c1-lib:int-add k 1))
                ((> k
                  (f2c1-lib:int-add i
                    (f2c1-lib:int-sub
                      1)))
                  nil)
                (tagbody
                  (setf temp
                    (+ temp
                      (*
                        (f2c1-lib:fref a-%data%
                          (k i)
                          ((1 lda) (1 *))
                          a-%offset%)
                        (f2c1-lib:fref b-%data%
                          (k j)
                          ((1 ldb$) (1 *))
                          b-%offset%))))))))))

```

```

(t
  (if nounit
    (setf temp
      (* temp
        (f2cl-lib:dconjg
          (f2cl-lib:fref a-%data%
            (i i)
            ((1 lda) (1 *))
            a-%offset%))))))
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
      (> k
        (f2cl-lib:int-add i
          (f2cl-lib:int-sub
            1)))
        nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (k i)
                ((1 lda) (1 *))
                a-%offset%))
            (f2cl-lib:fref b-%data%
              (k j)
              ((1 ldb$) (1 *))
              b-%offset%))))))
      (setf (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
        (* alpha temp))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf temp
        (f2cl-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *))
          b-%offset%))
      (cond
        (noconj
          (if nounit
            (setf temp
              (* temp

```

```

(f2c1-lib:fref a-%data%
  (i i)
  ((1 lda) (1 *))
  a-%offset%)))
(f2c1-lib:fdo (k (f2c1-lib:int-add i 1)
  (f2c1-lib:int-add k 1))
  (> k m) nil)
(tagbody
  (setf temp
    (+ temp
      (*
        (f2c1-lib:fref a-%data%
          (k i)
          ((1 lda) (1 *))
          a-%offset%)
        (f2c1-lib:fref b-%data%
          (k j)
          ((1 ldb$) (1 *))
          b-%offset%))))))
(t
  (if nunit
    (setf temp
      (* temp
        (f2c1-lib:dconjg
          (f2c1-lib:fref a-%data%
            (i i)
            ((1 lda) (1 *))
            a-%offset%))))))
    (f2c1-lib:fdo (k (f2c1-lib:int-add i 1)
      (f2c1-lib:int-add k 1))
      (> k m) nil)
    (tagbody
      (setf temp
        (+ temp
          (*
            (f2c1-lib:dconjg
              (f2c1-lib:fref a-%data%
                (k i)
                ((1 lda) (1 *))
                a-%offset%)
              (f2c1-lib:fref b-%data%
                (k j)
                ((1 ldb$) (1 *))
                b-%offset%))))))
          (setf (f2c1-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)
            (* alpha temp))))))
  (t

```

```
(cond
  ((char-equal transa #\N)
   (cond
    (upper
     (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
       ((> j 1) nil)
      (tagbody
       (setf temp alpha)
       (if nunit
        (setf temp
         (* temp
          (f2cl-lib:fref a-%data%
                        (j j)
                        ((1 lda) (1 *))
                        a-%offset%))))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
         (tagbody
          (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
                (* temp
                 (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%))))
          (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
            ((> k
              (f2cl-lib:int-add j
                (f2cl-lib:int-sub 1)))
              nil)
           (tagbody
            (cond
             ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
              (setf temp
               (* alpha
                (f2cl-lib:fref a-%data%
                              (k j)
                              ((1 lda) (1 *))
                              a-%offset%)))
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i m) nil)
               (tagbody
                (setf (f2cl-lib:fref b-%data%
                                    (i j)
                                    ((1 ldb$) (1 *))
                                    b-%offset%)
                      (+
                       (f2cl-lib:fref b-%data%
```

```

                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                                (* temp
                                (f2cl-lib:fref b-%data%
                                (i k)
                                ((1 ldb$) (1 *))
                                b-%offset%))))))))))
(t
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf temp alpha)
  (if nunit
    (setf temp
      (* temp
        (f2cl-lib:fref a-%data%
          (j j)
          ((1 lda) (1 *))
          a-%offset%)))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
        (* temp
          (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%))))
      (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
        (f2cl-lib:int-add k 1))
        (> k n) nil)
      (tagbody
        (cond
          ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
            (setf temp
              (* alpha
                (f2cl-lib:fref a-%data%
                  (k j)
                  ((1 lda) (1 *))
                  a-%offset%)))
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              (> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))

```



```

                                b-%offset%)
      (+
        (f2c1-lib:fref b-%data%
          (i j)
          ((1 ldb$) (1 *)))
        b-%offset%)
      (* temp
        (f2c1-lib:fref b-%data%
          (i k)
          ((1 ldb$) (1 *)))
        b-%offset%)))))))))))))
(t
  (cond
    (upper
      (f2c1-lib:fdo (k 1 (f2c1-lib:int-add k 1))
        ((> k n) nil)
      (tagbody
        (f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
          ((> j
            (f2c1-lib:int-add k
              (f2c1-lib:int-sub 1)))
            nil)
          (tagbody
            (cond
              ((/= (f2c1-lib:fref a (j k) ((1 lda) (1 *))) zero)
                (cond
                  (noconj
                    (setf temp
                      (* alpha
                        (f2c1-lib:fref a-%data%
                          (j k)
                          ((1 lda) (1 *)))
                        a-%offset%))))
                  (t
                    (setf temp
                      (* alpha
                        (f2c1-lib:dconjg
                          (f2c1-lib:fref a-%data%
                            (j k)
                            ((1 lda) (1 *)))
                          a-%offset%))))))
                (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
                  ((> i m) nil)
                  (tagbody
                    (setf (f2c1-lib:fref b-%data%
                      (i j)
                      ((1 ldb$) (1 *)))
                      b-%offset%)
                    (+
                      (f2c1-lib:fref b-%data%

```

```

                                (i j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                                (* temp
                                (f2cl-lib:fref b-%data%
                                (i k)
                                ((1 ldb$) (1 *))
                                b-%offset%)))))))))
(setf temp alpha)
(cond
  (nounit
    (cond
      (noconj
        (setf temp
          (* temp
            (f2cl-lib:fref a-%data%
              (k k)
              ((1 lda) (1 *))
              a-%offset%))))))
      (t
        (setf temp
          (* temp
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                (k k)
                ((1 lda) (1 *))
                a-%offset%)))))))))
(cond
  ((/= temp one)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
        (i k)
        ((1 ldb$) (1 *))
        b-%offset%)
        (* temp
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%))))))
  (t
    (f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
      (> k 1) nil)
    (tagbody
      (f2cl-lib:fdo (j (f2cl-lib:int-add k 1)
        (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (cond

```

```

( (/ = (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
  (cond
    (noconj
      (setf temp
        (* alpha
          (f2cl-lib:fref a-%data%
                        (j k)
                        ((1 lda) (1 *))
                        a-%offset%))))
      (t
        (setf temp
          (* alpha
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                            (j k)
                            ((1 lda) (1 *))
                            a-%offset%))))))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
            (+
              (f2cl-lib:fref b-%data%
                            (i j)
                            ((1 ldb$) (1 *))
                            b-%offset%)
              (* temp
                (f2cl-lib:fref b-%data%
                              (i k)
                              ((1 ldb$) (1 *))
                              b-%offset%))))))
      (setf temp alpha)
      (cond
        (nounit
          (cond
            (noconj
              (setf temp
                (* temp
                  (f2cl-lib:fref a-%data%
                                (k k)
                                ((1 lda) (1 *))
                                a-%offset%))))
              (t
                (setf temp
                  (* temp
                    (f2cl-lib:dconjg
                      (f2cl-lib:fref a-%data%

```

```

(k k)
((1 lda) (1 *))
a-%offset%)))))))))

(cond
  ((/= temp one)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     ((> i m) nil)
     (tagbody
      (setf (f2cl-lib:fref b-%data%
                          (i k)
                          ((1 ldb$) (1 *))
                          b-%offset%))
        (* temp
          (f2cl-lib:fref b-%data%
                        (i k)
                        ((1 ldb$) (1 *))
                        b-%offset%))))))))))

end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))))

```

ztrsm BLAS

— ztrsm.input —

```

)set break resume
)sys rm -f ztrsm.output
)spool ztrsm.output
)set message test on
)set message auto off
)clear all

```

```

)spool
)lisp (bye)

```

— ztrsm.help —

```

=====
ztrsm examples
=====
=====

```

Man Page Details

=====

NAME

ZTRSM - solve one of the matrix equations $\text{op}(A)X = \alpha B$, or $X\text{op}(A) = \alpha B$,

SYNOPSIS

SUBROUTINE ZTRSM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
LDA, B, LDB)

CHARACTER*1 SIDE, UPLO, TRANSA, DIAG

INTEGER M, N, LDA, LDB

COMPLEX*16 ALPHA

COMPLEX*16 A(LDA, *), B(LDB, *)

PURPOSE

ZTRSM solves one of the matrix equations

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A)$ is one of

$\text{op}(A) = A$ or $\text{op}(A) = A'$ or $\text{op}(A) = \text{conj}(A')$.

The matrix X is overwritten on B.

PARAMETERS

SIDE - CHARACTER*1.

On entry, SIDE specifies whether $\text{op}(A)$ appears on the left or right of X as follows:

SIDE = 'L' or 'l' $\text{op}(A)X = \alpha B$.

SIDE = 'R' or 'r' $X\text{op}(A) = \alpha B$.

Unchanged on exit.

UPLO - CHARACTER*1.

On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of $\text{op}(A)$ to be used in the matrix

multiplication as follows:

TRANSA = 'N' or 'n' $\text{op}(A) = A$.

TRANSA = 'T' or 't' $\text{op}(A) = A'$.

TRANSA = 'C' or 'c' $\text{op}(A) = \text{conjg}(A')$.

Unchanged on exit.

DIAG - CHARACTER*1.

On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA - COMPLEX*16 .

On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

is m

A

-
COMPLEX*16 array of DIMENSION (LDA, k), where k when SIDE = 'L' or 'l' and is n when SIDE = 'R' or 'r'. Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A

must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity. Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least $\max(1, m)$, when SIDE = 'R' or 'r' then LDA must be at least $\max(1, n)$. Unchanged on exit.

B - COMPLEX*16 array of DIMENSION (LDB, n).
Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

LDB - INTEGER.

On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least $\max(1, m)$. Unchanged on exit.

— ztrsm.f —

```

SUBROUTINE ZTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA,
$               B, LDB )
*   .. Scalar Arguments ..
CHARACTER*1    SIDE, UPLO, TRANSA, DIAG
INTEGER        M, N, LDA, LDB
COMPLEX*16     ALPHA
*   .. Array Arguments ..
COMPLEX*16     A( LDA, * ), B( LDB, * )
*   ..
*
*   Level 3 Blas routine.
*
*   -- Written on 8-February-1989.
*   Jack Dongarra, Argonne National Laboratory.
*   Iain Duff, AERE Harwell.
*   Jeremy Du Croz, Numerical Algorithms Group Ltd.
*   Sven Hammarling, Numerical Algorithms Group Ltd.
*
*

```

```

*      .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL         LSAME
*      .. External Subroutines ..
      EXTERNAL         XERBLA
*      .. Intrinsic Functions ..
      INTRINSIC         DCONJG, MAX
*      .. Local Scalars ..
      LOGICAL          LSIDE, NOCONJ, NOUNIT, UPPER
      INTEGER          I, INFO, J, K, NROWA
      COMPLEX*16       TEMP
*      .. Parameters ..
      COMPLEX*16       ONE
      PARAMETER         ( ONE = ( 1.0D+0, 0.0D+0 ) )
      COMPLEX*16       ZERO
      PARAMETER         ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      LSIDE = LSAME( SIDE , 'L' )
      IF( LSIDE )THEN
          NROWA = M
      ELSE
          NROWA = N
      END IF
      NOCONJ = LSAME( TRANSA, 'T' )
      NOUNIT = LSAME( DIAG , 'N' )
      UPPER = LSAME( UPLO , 'U' )
*
      INFO = 0
      IF( ( .NOT.LSIDE ) )AND.
$      ( .NOT.LSAME( SIDE , 'R' ) ) )THEN
          INFO = 1
      ELSE IF( ( .NOT.UPPER ) )AND.
$      ( .NOT.LSAME( UPLO , 'L' ) ) )THEN
          INFO = 2
      ELSE IF( ( .NOT.LSAME( TRANSA, 'N' ) ) )AND.
$      ( .NOT.LSAME( TRANSA, 'T' ) ) )AND.
$      ( .NOT.LSAME( TRANSA, 'C' ) ) )THEN
          INFO = 3
      ELSE IF( ( .NOT.LSAME( DIAG , 'U' ) ) )AND.
$      ( .NOT.LSAME( DIAG , 'N' ) ) )THEN
          INFO = 4
      ELSE IF( M .LT.0 )THEN
          INFO = 5
      ELSE IF( N .LT.0 )THEN
          INFO = 6
      ELSE IF( LDA.LT.MAX( 1, NROWA ) )THEN

```



```

        INFO = 9
      ELSE IF( LDB.LT.MAX( 1, M      ) )THEN
        INFO = 11
      END IF
      IF( INFO.NE.0 )THEN
        CALL XERBLA( 'ZTRSM ', INFO )
        RETURN
      END IF
*
*   Quick return if possible.
*
      IF( N.EQ.0 )
$      RETURN
*
*   And when  alpha.eq.zero.
*
      IF( ALPHA.EQ.ZERO )THEN
        DO 20, J = 1, N
          DO 10, I = 1, M
            B( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
          RETURN
        END IF
*
*   Start the operations.
*
      IF( LSIDE )THEN
        IF( LSAME( TRANSA, 'N' ) )THEN
*
*           Form  B := alpha*inv( A )*B.
*
          IF( UPPER )THEN
            DO 60, J = 1, N
              IF( ALPHA.NE.ONE )THEN
                DO 30, I = 1, M
                  B( I, J ) = ALPHA*B( I, J )
30                CONTINUE
              END IF
              DO 50, K = M, 1, -1
                IF( B( K, J ).NE.ZERO )THEN
                  IF( NOUNIT )
$                    B( K, J ) = B( K, J )/A( K, K )
                  DO 40, I = 1, K - 1
                    B( I, J ) = B( I, J ) - B( K, J )*A( I, K )
40                  CONTINUE
                END IF
              CONTINUE
50            CONTINUE
60          CONTINUE
          ELSE

```

```

DO 100, J = 1, N
  IF( ALPHA.NE.ONE )THEN
    DO 70, I = 1, M
      B( I, J ) = ALPHA*B( I, J )
70    CONTINUE
    END IF
    DO 90 K = 1, M
      IF( B( K, J ).NE.ZERO )THEN
        IF( NOUNIT )
          $      B( K, J ) = B( K, J )/A( K, K )
          DO 80, I = K + 1, M
            B( I, J ) = B( I, J ) - B( K, J )*A( I, K )
80          CONTINUE
          END IF
90        CONTINUE
100      CONTINUE
    END IF
  ELSE
*
*      Form B := alpha*inv( A' )*B
*      or   B := alpha*inv( conjg( A' ) )*B.
*
    IF( UPPER )THEN
      DO 140, J = 1, N
        DO 130, I = 1, M
          TEMP = ALPHA*B( I, J )
          IF( NOCONJ )THEN
            DO 110, K = 1, I - 1
              TEMP = TEMP - A( K, I )*B( K, J )
110          CONTINUE
          IF( NOUNIT )
            $      TEMP = TEMP/A( I, I )
          ELSE
            DO 120, K = 1, I - 1
              TEMP = TEMP - DCONJG( A( K, I ) )*B( K, J )
120          CONTINUE
          IF( NOUNIT )
            $      TEMP = TEMP/DCONJG( A( I, I ) )
          END IF
          B( I, J ) = TEMP
130        CONTINUE
140      CONTINUE
    ELSE
      DO 180, J = 1, N
        DO 170, I = M, 1, -1
          TEMP = ALPHA*B( I, J )
          IF( NOCONJ )THEN
            DO 150, K = I + 1, M
              TEMP = TEMP - A( K, I )*B( K, J )
150          CONTINUE

```

```

                IF( NOUNIT )
$                TEMP = TEMP/A( I, I )
                ELSE
                DO 160, K = I + 1, M
                    TEMP = TEMP - DCONJG( A( K, I ) ) * B( K, J )
160                CONTINUE
                IF( NOUNIT )
$                TEMP = TEMP/DCONJG( A( I, I ) )
                END IF
                B( I, J ) = TEMP
170                CONTINUE
180                CONTINUE
            END IF
        END IF
    ELSE
        IF( LSAME( TRANSA, 'N' ) ) THEN
*
*          Form B := alpha*B*inv( A ).
*
            IF( UPPER ) THEN
                DO 230, J = 1, N
                    IF( ALPHA.NE.ONE ) THEN
                        DO 190, I = 1, M
                            B( I, J ) = ALPHA * B( I, J )
190                        CONTINUE
                        END IF
                        DO 210, K = 1, J - 1
                            IF( A( K, J ).NE.ZERO ) THEN
                                DO 200, I = 1, M
                                    B( I, J ) = B( I, J ) - A( K, J ) * B( I, K )
200                                CONTINUE
                                END IF
                            CONTINUE
                        IF( NOUNIT ) THEN
                            TEMP = ONE/A( J, J )
                            DO 220, I = 1, M
                                B( I, J ) = TEMP * B( I, J )
220                            CONTINUE
                        END IF
                    CONTINUE
230                ELSE
                    DO 280, J = N, 1, -1
                        IF( ALPHA.NE.ONE ) THEN
                            DO 240, I = 1, M
                                B( I, J ) = ALPHA * B( I, J )
240                            CONTINUE
                        END IF
                        DO 260, K = J + 1, N
                            IF( A( K, J ).NE.ZERO ) THEN
                                DO 250, I = 1, M

```

```

                B( I, J ) = B( I, J ) - A( K, J ) * B( I, K )
250            CONTINUE
            END IF
260        CONTINUE
        IF( NOUNIT ) THEN
            TEMP = ONE / A( J, J )
            DO 270, I = 1, M
                B( I, J ) = TEMP * B( I, J )
270            CONTINUE
            END IF
280        CONTINUE
        END IF
    ELSE
*
*       Form B := alpha*B*inv( A' )
*       or   B := alpha*B*inv( conjg( A' ) ).
*
        IF( UPPER ) THEN
            DO 330, K = N, 1, -1
                IF( NOUNIT ) THEN
                    IF( NOCONJ ) THEN
                        TEMP = ONE / A( K, K )
                    ELSE
                        TEMP = ONE / DCONJG( A( K, K ) )
                    END IF
                    DO 290, I = 1, M
                        B( I, K ) = TEMP * B( I, K )
290                CONTINUE
            END IF
            DO 310, J = 1, K - 1
                IF( A( J, K ).NE.ZERO ) THEN
                    IF( NOCONJ ) THEN
                        TEMP = A( J, K )
                    ELSE
                        TEMP = DCONJG( A( J, K ) )
                    END IF
                    DO 300, I = 1, M
                        B( I, J ) = B( I, J ) - TEMP * B( I, K )
300                CONTINUE
            END IF
310        CONTINUE
        IF( ALPHA.NE.ONE ) THEN
            DO 320, I = 1, M
                B( I, K ) = ALPHA * B( I, K )
320            CONTINUE
        END IF
330    CONTINUE
    ELSE
        DO 380, K = 1, N
            IF( NOUNIT ) THEN

```

```

        IF( NOCONJ )THEN
            TEMP = ONE/A( K, K )
        ELSE
            TEMP = ONE/DCONJG( A( K, K ) )
        END IF
        DO 340, I = 1, M
            B( I, K ) = TEMP*B( I, K )
340        CONTINUE
    END IF
    DO 360, J = K + 1, N
        IF( A( J, K ).NE.ZERO )THEN
            IF( NOCONJ )THEN
                TEMP = A( J, K )
            ELSE
                TEMP = DCONJG( A( J, K ) )
            END IF
            DO 350, I = 1, M
                B( I, J ) = B( I, J ) - TEMP*B( I, K )
350            CONTINUE
        END IF
360    CONTINUE
        IF( ALPHA.NE.ONE )THEN
            DO 370, I = 1, M
                B( I, K ) = ALPHA*B( I, K )
370            CONTINUE
        END IF
380    CONTINUE
    END IF
END IF
END IF
*
    RETURN
*
*   End of ZTRSM .
*
    END

```

— BLAS 3 ztrsm —

```

(let* ((one (complex 1.0 0.0)) (zero (complex 0.0 0.0)))
  (declare (type (complex double-float) one) (type (complex double-float) zero))
  (defun ztrsm (side uplo transa diag m n alpha a lda b ldb$)
    (declare (type (simple-array (complex double-float) (*)) b a)
              (type (complex double-float) alpha)
              (type fixnum ldb$ lda n m)
              (type character diag transa uplo side))

```

```

(f2cl-lib:with-multi-array-data
  ((side character side-%data% side-%offset%)
   (uplo character uplo-%data% uplo-%offset%)
   (transa character transa-%data% transa-%offset%)
   (diag character diag-%data% diag-%offset%)
   (a (complex double-float) a-%data% a-%offset%)
   (b (complex double-float) b-%data% b-%offset%))
  (prog ((temp #C(0.0 0.0)) (i 0) (info 0) (j 0) (k 0) (nrowa 0)
        (lside nil) (noconj nil) (nounit nil) (upper nil))
    (declare (type (complex double-float) temp)
              (type fixnum i info j k nrowa)
              (type (member t nil) lside noconj nounit upper))
    (setf lside (char-equal side #\L))
    (cond
      (lside
       (setf nrowa m))
      (t
       (setf nrowa n)))
    (setf noconj (char-equal transa #\T))
    (setf nounit (char-equal diag #\N))
    (setf upper (char-equal uplo #\U))
    (setf info 0)
    (cond
      ((and (not lside) (not (char-equal side #\R)))
       (setf info 1))
      ((and (not upper) (not (char-equal uplo #\L)))
       (setf info 2))
      ((and (not (char-equal transa #\N))
             (not (char-equal transa #\T))
             (not (char-equal transa #\C)))
       (setf info 3))
      ((and (not (char-equal diag #\U)) (not (char-equal diag #\N)))
       (setf info 4))
      ((< m 0)
       (setf info 5))
      ((< n 0)
       (setf info 6))
      ((< lda (max (the fixnum 1) (the fixnum nrowa)))
       (setf info 9))
      ((< ldb$ (max (the fixnum 1) (the fixnum m)))
       (setf info 11)))
    (cond
      ((/= info 0)
       (error
        "** On entry to ~a parameter number ~a had an illegal value~%"
        "ZTRSM" info)
       (go end_label)))
    (if (= n 0) (go end_label))
    (cond
      ((= alpha zero)

```



```

                                b-%offset%)
(f2cl-lib:fref a-%data%
  (k k)
  ((1 lda) (1 *))
  a-%offset%)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (f2cl-lib:int-add k
      (f2cl-lib:int-sub
        1)))
    nil)
(tagbody
  (setf (f2cl-lib:fref b-%data%
    (i j)
    ((1 ldb$) (1 *))
    b-%offset%)
    (-
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)
      (*
        (f2cl-lib:fref b-%data%
          (k j)
          ((1 ldb$) (1 *))
          b-%offset%)
        (f2cl-lib:fref a-%data%
          (i k)
          ((1 lda) (1 *))
          a-%offset%)))))))))
(t
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref b-%data%
            (i j)
            ((1 ldb$) (1 *))
            b-%offset%)
            (* alpha
              (f2cl-lib:fref b-%data%
                (i j)
                ((1 ldb$) (1 *))
                b-%offset%))))))
        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
          (> k m) nil)

```



```

(tagbody
  (cond
    ((/= (f2cl-lib:fref b (k j) ((1 ldb$) (1 *))) zero)
      (if nounit
        (setf (f2cl-lib:fref b-%data%
                              (k j)
                              ((1 ldb$) (1 *))
                              b-%offset%))
              (/
                (f2cl-lib:fref b-%data%
                              (k j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
                (f2cl-lib:fref a-%data%
                              (k k)
                              ((1 lda) (1 *))
                              a-%offset%))))
      (f2cl-lib:fdo (i (f2cl-lib:int-add k 1)
                    (f2cl-lib:int-add i 1))
                    (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%))
              (-
                (f2cl-lib:fref b-%data%
                              (i j)
                              ((1 ldb$) (1 *))
                              b-%offset%)
                (*
                  (f2cl-lib:fref b-%data%
                                (k j)
                                ((1 ldb$) (1 *))
                                b-%offset%)
                  (f2cl-lib:fref a-%data%
                                (i k)
                                ((1 lda) (1 *))
                                a-%offset%))))))))))
(t
  (cond
    (upper
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                    (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      (> i m) nil)
        (tagbody
          (setf temp
                (* alpha

```

```
(f2cl-lib:fref b-%data%
              (i j)
              ((1 ldb$) (1 *)))
b-%offset%)))

(cond
(noconj
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
               (> k
                (f2cl-lib:int-add i
                                   (f2cl-lib:int-sub
                                    1))))
      nil)
(tagbody
(setf temp
(- temp
(*
(f2cl-lib:fref a-%data%
                (k i)
                ((1 lda) (1 *))
a-%offset%)
(f2cl-lib:fref b-%data%
                (k j)
                ((1 ldb$) (1 *))
b-%offset%))))))

(if nounit
(setf temp
(/ temp
(f2cl-lib:fref a-%data%
                (i i)
                ((1 lda) (1 *))
a-%offset%))))))

(t
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
               (> k
                (f2cl-lib:int-add i
                                   (f2cl-lib:int-sub
                                    1))))
      nil)
(tagbody
(setf temp
(- temp
(*
(f2cl-lib:dconjg
(f2cl-lib:fref a-%data%
                (k i)
                ((1 lda) (1 *))
a-%offset%)
(f2cl-lib:fref b-%data%
                (k j)
                ((1 ldb$) (1 *))
```

```

                                                    b-%offset%))))))
    (if nounit
      (setf temp
        (/ temp
          (f2cl-lib:dconjg
            (f2cl-lib:fref a-%data%
                          (i i)
                          ((1 lda) (1 *))
                          a-%offset%))))))
    (setf (f2cl-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *))
                        b-%offset%)
      temp))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
 (tagbody
  (f2cl-lib:fdo (i m
                (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
   (> i 1) nil)
  (tagbody
   (setf temp
     (* alpha
       (f2cl-lib:fref b-%data%
                     (i j)
                     ((1 ldb$) (1 *))
                     b-%offset%)))
   (cond
    (noconj
     (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                     (f2cl-lib:int-add k 1))
      (> k m) nil)
     (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:fref a-%data%
                          (k i)
                          ((1 lda) (1 *))
                          a-%offset%)
            (f2cl-lib:fref b-%data%
                          (k j)
                          ((1 ldb$) (1 *))
                          b-%offset%))))))
     (if nounit
       (setf temp
         (/ temp
           (f2cl-lib:fref a-%data%
                         (i i)

```

```

((1 lda) (1 *))
a-%offset%))))))

(t
  (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
                  (f2cl-lib:int-add k 1))
                (> k m) nil)
    (tagbody
      (setf temp
        (- temp
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                            (k i)
                            ((1 lda) (1 *))
                            a-%offset%))
            (f2cl-lib:fref b-%data%
                            (k j)
                            ((1 ldb$) (1 *))
                            b-%offset%))))))
      (if nunit
        (setf temp
          (/ temp
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data%
                            (i i)
                            ((1 lda) (1 *))
                            a-%offset%))))))
      (setf (f2cl-lib:fref b-%data%
                          (i j)
                          ((1 ldb$) (1 *))
                          b-%offset%)
            temp))))))

(t
  (cond
    ((char-equal transa #\N)
      (cond
        (upper
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                        (> j n) nil)
            (tagbody
              (cond
                ((/= alpha one)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                                (> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref b-%data%
                                            (i j)
                                            ((1 ldb$) (1 *))
                                            b-%offset%)
                            (* alpha

```

```
(f2c1-lib:fref b-%data%
              (i j)
              ((1 ldb$) (1 *))
              b-%offset%))))))
(f2c1-lib:fdo (k 1 (f2c1-lib:int-add k 1))
              (> k
               (f2c1-lib:int-add j
                                   (f2c1-lib:int-sub 1)))
              nil)
(tagbody
 (cond
  ((/= (f2c1-lib:fref a (k j) ((1 lda) (1 *))) zero)
   (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
                  (> i m) nil)
   (tagbody
    (setf (f2c1-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *))
                        b-%offset%)
          (-
            (f2c1-lib:fref b-%data%
                            (i j)
                            ((1 ldb$) (1 *))
                            b-%offset%)
            (*
             (f2c1-lib:fref a-%data%
                             (k j)
                             ((1 lda) (1 *))
                             a-%offset%)
             (f2c1-lib:fref b-%data%
                             (i k)
                             ((1 ldb$) (1 *))
                             b-%offset%))))))))))
(cond
 (nunit
  (setf temp
        (/ one
           (f2c1-lib:fref a-%data%
                           (j j)
                           ((1 lda) (1 *))
                           a-%offset%)))
  (f2c1-lib:fdo (i 1 (f2c1-lib:int-add i 1))
                 (> i m) nil)
  (tagbody
   (setf (f2c1-lib:fref b-%data%
                       (i j)
                       ((1 ldb$) (1 *))
                       b-%offset%)
         (* temp
            (f2c1-lib:fref b-%data%
```

```

(i j)
((1 ldb$) (1 *))
b-%offset%)))))))))
(t
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j 1) nil)
  (tagbody
    (cond
      ((/= alpha one)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref b-%data%
              (i j)
              ((1 ldb$) (1 *))
              b-%offset%)
              (* alpha
                (f2cl-lib:fref b-%data%
                  (i j)
                  ((1 ldb$) (1 *))
                  b-%offset%))))))
        (f2cl-lib:fdo (k (f2cl-lib:int-add j 1)
          (f2cl-lib:int-add k 1))
            (> k n) nil)
            (tagbody
              (cond
                ((/= (f2cl-lib:fref a (k j) ((1 lda) (1 *))) zero)
                  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    (> i m) nil)
                    (tagbody
                      (setf (f2cl-lib:fref b-%data%
                        (i j)
                        ((1 ldb$) (1 *))
                        b-%offset%)
                        (-
                          (f2cl-lib:fref b-%data%
                            (i j)
                            ((1 ldb$) (1 *))
                            b-%offset%)
                          (*
                            (f2cl-lib:fref a-%data%
                              (k j)
                              ((1 lda) (1 *))
                              a-%offset%)
                            (f2cl-lib:fref b-%data%
                              (i k)
                              ((1 ldb$) (1 *))
                              b-%offset%))))))))
                (t
                  (tagbody
                    (setf (f2cl-lib:fref b-%data%
                      (i j)
                      ((1 ldb$) (1 *))
                      b-%offset%)
                      (*
                        (f2cl-lib:fref a-%data%
                          (k j)
                          ((1 lda) (1 *))
                          a-%offset%)
                        (f2cl-lib:fref b-%data%
                          (i k)
                          ((1 ldb$) (1 *))
                          b-%offset%))))))))
              (cond
                (nunit

```



```

(f2cl-lib:fref b-%data%
 (i k)
 ((1 ldb$) (1 *))
 b-%offset%))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j
  (f2cl-lib:int-add k
   (f2cl-lib:int-sub 1)))
  nil)
(tagbody
 (cond
  ((/= (f2cl-lib:fref a (j k) ((1 lda) (1 *))) zero)
   (cond
    (noconj
     (setf temp
      (f2cl-lib:fref a-%data%
       (j k)
       ((1 lda) (1 *))
       a-%offset%)))
     (t
      (setf temp
       (coerce
        (f2cl-lib:dconjg
         (f2cl-lib:fref a-%data%
          (j k)
          ((1 lda) (1 *))
          a-%offset%))
        '(complex double-float))))))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
   (tagbody
    (setf (f2cl-lib:fref b-%data%
     (i j)
     ((1 ldb$) (1 *))
     b-%offset%)
     (-
      (f2cl-lib:fref b-%data%
       (i j)
       ((1 ldb$) (1 *))
       b-%offset%)
      (* temp
       (f2cl-lib:fref b-%data%
        (i k)
        ((1 ldb$) (1 *))
        b-%offset%)))))))))
(cond
 ((/= alpha one)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
   (> i m) nil)
 (tagbody

```



```

(noconj
  (setf temp
    (f2cl-lib:fref a-%data%
      (j k)
      ((1 lda) (1 *))
      a-%offset%)))

(t
  (setf temp
    (coerce
      (f2cl-lib:dconjg
        (f2cl-lib:fref a-%data%
          (j k)
          ((1 lda) (1 *))
          a-%offset%))
      '(complex double-float))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)

(tagbody
  (setf (f2cl-lib:fref b-%data%
    (i j)
    ((1 ldb$) (1 *))
    b-%offset%)

    (-
      (f2cl-lib:fref b-%data%
        (i j)
        ((1 ldb$) (1 *))
        b-%offset%)

      (* temp
        (f2cl-lib:fref b-%data%
          (i k)
          ((1 ldb$) (1 *))
          b-%offset%))))))

(cond
  ((/= alpha one)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data%
        (i k)
        ((1 ldb$) (1 *))
        b-%offset%)

        (* alpha
          (f2cl-lib:fref b-%data%
            (i k)
            ((1 ldb$) (1 *))
            b-%offset%))))))

end_label
  (return (values nil nil nil nil nil nil nil nil nil nil)))

```

Chapter 6

LAPACK

dbdsdc LAPACK

— dbdsdc.input —

```
)set break resume
)sys rm -f dbdsdc.output
)spool dbdsdc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dbdsdc.help —

```
=====
dbdsdc examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DBDSDC - the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B

SYNOPSIS

```

SUBROUTINE DBDSDC( UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ, WORK,
                  IWORK, INFO )

      CHARACTER      COMPQ, UPLO

      INTEGER        INFO, LDU, LDVT, N

      INTEGER        IQ( * ), IWORK( * )

      DOUBLE         PRECISION  D( * ), E( * ), Q( * ), U( LDU, * ), VT(
                  LDVT, * ), WORK( * )

```

PURPOSE

DBDSDC computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B: $B = U * S * VT$, using a divide and conquer method, where S is a diagonal matrix with non-negative diagonal elements (the singular values of B), and U and VT are orthogonal matrices of left and right singular vectors, respectively. DBDSDC can be used to compute all singular values, and optionally, singular vectors or singular vectors in compact form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none. See DLASD3 for details.

The code currently calls DLASDQ if singular values only are desired. However, it can be slightly modified to compute singular values using the divide and conquer method.

ARGUMENTS

```

UPLO      (input) CHARACTER*1
          = 'U': B is upper bidiagonal.
          = 'L': B is lower bidiagonal.

COMPQ     (input) CHARACTER*1
          Specifies whether singular vectors are to be computed as follows:
          = 'N': Compute singular values only;
          = 'P': Compute singular values and compute singular vectors in compact form;
          = 'I': Compute singular values and singular vectors.

N         (input) INTEGER
          The order of the matrix B.  N >= 0.

```

- D** (input/output) DOUBLE PRECISION array, dimension (N)
On entry, the n diagonal elements of the bidiagonal matrix B.
On exit, if INFO=0, the singular values of B.
- E** (input/output) DOUBLE PRECISION array, dimension (N-1)
On entry, the elements of E contain the offdiagonal elements of the bidiagonal matrix whose SVD is desired. On exit, E has been destroyed.
- U** (output) DOUBLE PRECISION array, dimension (LDU,N)
If COMPQ = 'I', then: On exit, if INFO = 0, U contains the left singular vectors of the bidiagonal matrix. For other values of COMPQ, U is not referenced.
- LDU** (input) INTEGER
The leading dimension of the array U. LDU >= 1. If singular vectors are desired, then LDU >= max(1, N).
- VT** (output) DOUBLE PRECISION array, dimension (LDVT,N)
If COMPQ = 'I', then: On exit, if INFO = 0, VT contains the right singular vectors of the bidiagonal matrix. For other values of COMPQ, VT is not referenced.
- LDVT** (input) INTEGER
The leading dimension of the array VT. LDVT >= 1. If singular vectors are desired, then LDVT >= max(1, N).
- Q** (output) DOUBLE PRECISION array, dimension (LDQ)
If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring $O(N \log N)$ space instead of $2*N**2$. In particular, Q contains all the DOUBLE PRECISION data in LDQ >= $N*(11 + 2*SMLSIZ + 8*INT(LOG_2(N/(SMLSIZ+1))))$ words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, Q is not referenced.
- IQ** (output) INTEGER array, dimension (LDIQ)
If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring $O(N \log N)$ space instead of $2*N**2$. In particular, IQ contains all INTEGER data in LDIQ >= $N*(3 + 3*INT(LOG_2(N/(SMLSIZ+1))))$ words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, IQ is not referenced.
- WORK** (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
If COMPQ = 'N' then LWORK >= (4 * N). If COMPQ = 'P' then LWORK >= (6 * N). If COMPQ = 'I' then LWORK >= (3 * N**2 + 4 *

N).

IWORK (workspace) INTEGER array, dimension (8*N)

INFO (output) INTEGER

= 0: successful exit.

< 0: if INFO = -i, the i-th argument had an illegal value.

> 0: The algorithm failed to compute an singular value. The update process of divide and conquer failed.

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of California at Berkeley, USA

The input arguments are:

- uplo - simple-array character (1)
- compq - (simple-array character (1)
- n - fixnum
- d - array doublefloat
- e - array doublefloat
- u - array doublefloat
- ldu - fixnum
- vt - doublefloat
- ldvt - fixnum
- q - array doublefloat
- iq - array fixnum
- work - array doublefloat
- iwork - array fixnum
- info - fixnum

The return values are:

- uplo - nil
- compq - nil
- n - nil
- d - nil
- e - nil
- u - nil
- ldu - nil
- vt - nil
- ldvt - nil
- q - nil
- iq - nil
- work - nil
- iwork - nil
- info - info

[dlasr p??]
 [dswap p??]
 [dlasda p??]
 [dlsd0 p??]
 [dlamch p??]
 [dlacl p??]
 [dlaust p??]
 [dlaset p??]
 [dlsdq p??]
 [dlartg p??]
 [dcopy p??]
 [ilaenv p??]
 [xerbla p??]
 [char-equal p??]

— dbdsdc.f —

```

      SUBROUTINE DBDSDC( UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ,
$                        WORK, IWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --

```

```

*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      December 1, 1999
*
*      .. Scalar Arguments ..
*      CHARACTER          COMPQ, UPLO
*      INTEGER            INFO, LDU, LDVT, N
*
*      ..
*      .. Array Arguments ..
*      INTEGER            IQ( * ), IWORK( * )
*      DOUBLE PRECISION   D( * ), E( * ), Q( * ), U( LDU, * ),
*      $                  VT( LDVT, * ), WORK( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION   ZERO, ONE, TWO
*      PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0 )
*
*      ..
*      .. Local Scalars ..
*      INTEGER            DIFL, DIFR, GIVCOL, GIVNUM, GIVPTR, I, IC,
*      $                  ICOMPQ, IERR, II, IS, IU, IUPLO, IVT, J, K, KK,
*      $                  MLVL, NM1, NSIZE, PERM, POLES, QSTART, SMLSIZ,
*      $                  SMLSZP, SQRE, START, WSTART, Z
*      DOUBLE PRECISION   CS, EPS, ORGNRM, P, R, SN
*
*      ..
*      .. External Functions ..
*      LOGICAL            LSAME
*      INTEGER            ILAENV
*      DOUBLE PRECISION   DLAMCH, DLANST
*      EXTERNAL           LSAME, ILAENV, DLAMCH, DLANST
*
*      ..
*      .. External Subroutines ..
*      EXTERNAL           DCOPY, DLARTG, DLASCL, DLASDO, DLASDA, DLASDQ,
*      $                  DLASET, DLASR, DSWAP, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC          ABS, DBLE, INT, LOG, SIGN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
*      INFO = 0
*
*      IUPLO = 0
*      IF( LSAME( UPLO, 'U' ) )
*      $   IUPLO = 1
*      IF( LSAME( UPLO, 'L' ) )

```

```

$   IUPLO = 2
   IF( LSAME( COMPQ, 'N' ) ) THEN
       ICOMPQ = 0
   ELSE IF( LSAME( COMPQ, 'P' ) ) THEN
       ICOMPQ = 1
   ELSE IF( LSAME( COMPQ, 'I' ) ) THEN
       ICOMPQ = 2
   ELSE
       ICOMPQ = -1
   END IF
   IF( IUPLO.EQ.0 ) THEN
       INFO = -1
   ELSE IF( ICOMPQ.LT.0 ) THEN
       INFO = -2
   ELSE IF( N.LT.0 ) THEN
       INFO = -3
   ELSE IF( ( LDU.LT.1 ) .OR. ( ( ICOMPQ.EQ.2 ) .AND. ( LDU.LT.
$       N ) ) ) THEN
       INFO = -7
   ELSE IF( ( LDVT.LT.1 ) .OR. ( ( ICOMPQ.EQ.2 ) .AND. ( LDVT.LT.
$       N ) ) ) THEN
       INFO = -9
   END IF
   IF( INFO.NE.0 ) THEN
       CALL XERBLA( 'DBDSDC', -INFO )
       RETURN
   END IF

*
*   Quick return if possible
*

   IF( N.EQ.0 )
$   RETURN
   SMLSIZ = ILAENV( 9, 'DBDSDC', ' ', 0, 0, 0, 0 )
   IF( N.EQ.1 ) THEN
       IF( ICOMPQ.EQ.1 ) THEN
           Q( 1 ) = SIGN( ONE, D( 1 ) )
           Q( 1+SMLSIZ*N ) = ONE
       ELSE IF( ICOMPQ.EQ.2 ) THEN
           U( 1, 1 ) = SIGN( ONE, D( 1 ) )
           VT( 1, 1 ) = ONE
       END IF
       D( 1 ) = ABS( D( 1 ) )
       RETURN
   END IF
   NM1 = N - 1

*
*   If matrix lower bidiagonal, rotate to be upper bidiagonal
*   by applying Givens rotations on the left
*

   WSTART = 1

```

```

QSTART = 3
IF( ICOMPQ.EQ.1 ) THEN
    CALL DCOPY( N, D, 1, Q( 1 ), 1 )
    CALL DCOPY( N-1, E, 1, Q( N+1 ), 1 )
END IF
IF( IUPLO.EQ.2 ) THEN
    QSTART = 5
    WSTART = 2*N - 1
    DO 10 I = 1, N - 1
        CALL DLARTG( D( I ), E( I ), CS, SN, R )
        D( I ) = R
        E( I ) = SN*D( I+1 )
        D( I+1 ) = CS*D( I+1 )
        IF( ICOMPQ.EQ.1 ) THEN
            Q( I+2*N ) = CS
            Q( I+3*N ) = SN
        ELSE IF( ICOMPQ.EQ.2 ) THEN
            WORK( I ) = CS
            WORK( NM1+I ) = -SN
        END IF
10    CONTINUE
    END IF
*
*   If ICOMPQ = 0, use DLASDQ to compute the singular values.
*
IF( ICOMPQ.EQ.0 ) THEN
    CALL DLASDQ( 'U', 0, N, 0, 0, 0, 0, D, E, VT, LDVT, U, LDU, U,
$           LDU, WORK( WSTART ), INFO )
    GO TO 40
END IF
*
*   If N is smaller than the minimum divide size SMLSIZ, then solve
*   the problem with another solver.
*
IF( N.LE.SMLSIZ ) THEN
    IF( ICOMPQ.EQ.2 ) THEN
        CALL DLASET( 'A', N, N, ZERO, ONE, U, LDU )
        CALL DLASET( 'A', N, N, ZERO, ONE, VT, LDVT )
        CALL DLASDQ( 'U', 0, N, N, N, 0, D, E, VT, LDVT, U, LDU, U,
$           LDU, WORK( WSTART ), INFO )
    ELSE IF( ICOMPQ.EQ.1 ) THEN
        IU = 1
        IVT = IU + N
        CALL DLASET( 'A', N, N, ZERO, ONE, Q( IU+( QSTART-1 )*N ),
$           N )
        CALL DLASET( 'A', N, N, ZERO, ONE, Q( IVT+( QSTART-1 )*N ),
$           N )
        CALL DLASDQ( 'U', 0, N, N, N, 0, D, E,
$           Q( IVT+( QSTART-1 )*N ), N,
$           Q( IU+( QSTART-1 )*N ), N,

```

```

$          Q( IU+( QSTART-1 )*N ), N, WORK( WSTART ),
$          INFO )
      END IF
      GO TO 40
END IF
*
IF( ICOMPQ.EQ.2 ) THEN
      CALL DLASET( 'A', N, N, ZERO, ONE, U, LDU )
      CALL DLASET( 'A', N, N, ZERO, ONE, VT, LDVT )
END IF
*
*      Scale.
*
      ORGNRM = DLANST( 'M', N, D, E )
      IF( ORGNRM.EQ.ZERO )
$      RETURN
      CALL DLASCL( 'G', 0, 0, ORGNRM, ONE, N, 1, D, N, IERR )
      CALL DLASCL( 'G', 0, 0, ORGNRM, ONE, NM1, 1, E, NM1, IERR )
*
      EPS = DLAMCH( 'Epsilon' )
*
      MLVL = INT( LOG( DBLE( N ) / DBLE( SMLSIZ+1 ) ) / LOG( TWO ) ) + 1
      SMLSZP = SMLSIZ + 1
*
      IF( ICOMPQ.EQ.1 ) THEN
            IU = 1
            IVT = 1 + SMLSIZ
            DIFL = IVT + SMLSZP
            DIFR = DIFL + MLVL
            Z = DIFR + MLVL*2
            IC = Z + MLVL
            IS = IC + 1
            POLES = IS + 1
            GIVNUM = POLES + 2*MLVL
*
            K = 1
            GIVPTR = 2
            PERM = 3
            GIVCOL = PERM + MLVL
      END IF
*
      DO 20 I = 1, N
            IF( ABS( D( I ) ).LT.EPS ) THEN
                  D( I ) = SIGN( EPS, D( I ) )
            END IF
20 CONTINUE
*
      START = 1
      SQRE = 0
*

```

```

DO 30 I = 1, NM1
  IF( ( ABS( E( I ) ) .LT. EPS ) .OR. ( I.EQ.NM1 ) ) THEN
*
*   Subproblem found. First determine its size and then
*   apply divide and conquer on it.
*
    IF( I.LT.NM1 ) THEN
*
*   A subproblem with E(I) small for I < NM1.
*
      NSIZE = I - START + 1
      ELSE IF( ABS( E( I ) ) .GE. EPS ) THEN
*
*   A subproblem with E(NM1) not too small but I = NM1.
*
      NSIZE = N - START + 1
      ELSE
*
*   A subproblem with E(NM1) small. This implies an
*   1-by-1 subproblem at D(N). Solve this 1-by-1 problem
*   first.
*
      NSIZE = I - START + 1
      IF( ICOMPQ.EQ.2 ) THEN
        U( N, N ) = SIGN( ONE, D( N ) )
        VT( N, N ) = ONE
      ELSE IF( ICOMPQ.EQ.1 ) THEN
        Q( N+( QSTART-1 )*N ) = SIGN( ONE, D( N ) )
        Q( N+( SMLSIZ+QSTART-1 )*N ) = ONE
      END IF
      D( N ) = ABS( D( N ) )
    END IF
    IF( ICOMPQ.EQ.2 ) THEN
      CALL DLASDO( NSIZE, SQRE, D( START ), E( START ),
$           U( START, START ), LDU, VT( START, START ),
$           LDVT, SMLSIZ, IWORK, WORK( WSTART ), INFO )
    ELSE
      CALL DLASDA( ICOMPQ, SMLSIZ, NSIZE, SQRE, D( START ),
$           E( START ), Q( START+( IU+QSTART-2 )*N ), N,
$           Q( START+( IVT+QSTART-2 )*N ),
$           IQ( START+K*N ), Q( START+( DIFL+QSTART-2 )*
$           N ), Q( START+( DIFR+QSTART-2 )*N ),
$           Q( START+( Z+QSTART-2 )*N ),
$           Q( START+( POLES+QSTART-2 )*N ),
$           IQ( START+GIVPTR*N ), IQ( START+GIVCOL*N ),
$           N, IQ( START+PERM*N ),
$           Q( START+( GIVNUM+QSTART-2 )*N ),
$           Q( START+( IC+QSTART-2 )*N ),
$           Q( START+( IS+QSTART-2 )*N ),
$           WORK( WSTART ), IWORK, INFO )

```

```

                IF( INFO.NE.0 ) THEN
                    RETURN
                END IF
            END IF
            START = I + 1
        END IF
30 CONTINUE
*
*   Unscale
*
        CALL DLASCL( 'G', 0, 0, ONE, ORGNRM, N, 1, D, N, IERR )
40 CONTINUE
*
*   Use Selection Sort to minimize swaps of singular vectors
*
        DO 60 II = 2, N
            I = II - 1
            KK = I
            P = D( I )
            DO 50 J = II, N
                IF( D( J ).GT.P ) THEN
                    KK = J
                    P = D( J )
                END IF
            CONTINUE
50         IF( KK.NE.I ) THEN
            D( KK ) = D( I )
            D( I ) = P
            IF( ICOMPQ.EQ.1 ) THEN
                IQ( I ) = KK
            ELSE IF( ICOMPQ.EQ.2 ) THEN
                CALL DSWAP( N, U( 1, I ), 1, U( 1, KK ), 1 )
                CALL DSWAP( N, VT( I, 1 ), LDVT, VT( KK, 1 ), LDVT )
            END IF
            ELSE IF( ICOMPQ.EQ.1 ) THEN
                IQ( I ) = I
            END IF
        CONTINUE
60 CONTINUE
*
*   If ICOMPQ = 1, use IQ(N,1) as the indicator for UPLO
*
*
        IF( ICOMPQ.EQ.1 ) THEN
            IF( IUPLD.EQ.1 ) THEN
                IQ( N ) = 1
            ELSE
                IQ( N ) = 0
            END IF
        END IF
*
*   If B is lower bidiagonal, update U by those Givens rotations

```

```

*      which rotated B to be upper bidiagonal
*
      IF( ( IUPLO.EQ.2 ) .AND. ( ICOMPQ.EQ.2 ) )
$     CALL DLASR( 'L', 'V', 'B', N, N, WORK( 1 ), WORK( N ), U, LDU )
*
      RETURN
*
*      End of DBDSDC
*
      END

```

— LAPACK dbdsdc —

```

(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two))
  (defun dbdsdc (uplo compq n d e u ldu vt ldvt q iq work iwork info)
    (declare (type (simple-array fixnum (*)) iwork iq)
              (type (simple-array double-float (*)) work q vt u e d)
              (type fixnum info ldvt ldu n)
              (type character compq uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (compq character compq-%data% compq-%offset%)
       (d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (u double-float u-%data% u-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (q double-float q-%data% q-%offset%)
       (work double-float work-%data% work-%offset%)
       (iq fixnum iq-%data% iq-%offset%)
       (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((cs 0.0) (eps 0.0) (orgnrm 0.0) (p 0.0) (r 0.0) (sn 0.0) (difl 0)
              (difr 0) (givcol 0) (givnum 0) (givptr 0) (i 0) (ic 0) (icompq 0)
              (ierr 0) (ii 0) (is 0) (iu 0) (iuplo 0) (ivt 0) (j 0) (k 0) (kk 0)
              (mlvl 0) (nm1 0) (nsize 0) (perm 0) (poles 0) (qstart 0)
              (smlsiz 0) (smlszp 0) (sqre 0) (start 0) (wstart 0) (z 0))
        (declare (type (double-float) cs eps orgnrm p r sn)
                  (type fixnum difl difr givcol givnum givptr i ic
                           icompq ierr ii is iu iuplo ivt j k
                           kk mlvl nm1 nsize perm poles qstart
                           smlsiz smlszp sqre start wstart z))

          (setf info 0)
          (setf iuplo 0)
          (if (char-equal uplo #\U) (setf iuplo 1))

```



```

(if (char-equal uplo #\L) (setf iuplo 2))
(cond
  ((char-equal compq #\N) (setf icompq 0))
  ((char-equal compq #\P) (setf icompq 1))
  ((char-equal compq #\I) (setf icompq 2))
  (t (setf icompq -1)))
(cond
  ((= iuplo 0) (setf info -1))
  ((< icompq 0) (setf info -2))
  ((< n 0) (setf info -3))
  ((or (< ldu 1) (and (= icompq 2) (< ldu n))) (setf info -7))
  ((or (< ldvt 1) (and (= icompq 2) (< ldvt n))) (setf info -9)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DBDSDC" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= n 0) (go end_label))
(setf smlsiz (ilaenv 9 "DBDSDC" " " 0 0 0 0))
(cond
  ((= n 1)
   (cond
    ((= icompq 1)
     (setf
      (f2cl-lib:fref q-%data% (1) ((1 *)) q-%offset%)
      (f2cl-lib:sign one
       (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
     (setf
      (f2cl-lib:fref q-%data%
       ((f2cl-lib:int-add 1
        (f2cl-lib:int-mul smlsiz n))) ((1 *)) q-%offset%)
      one))
    ((= icompq 2)
     (setf
      (f2cl-lib:fref u-%data% (1 1) ((1 ldu) (1 *)) u-%offset%)
      (f2cl-lib:sign one (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
     (setf
      (f2cl-lib:fref vt-%data% (1 1) ((1 ldvt) (1 *)) vt-%offset%)
      one))
     (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
            (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
     (go end_label)))
  (setf nm1 (f2cl-lib:int-sub n 1))
  (setf wstart 1)
  (setf qstart 3)
  (cond
    ((= icompq 1)
     (dcopy n 1 (f2cl-lib:array-slice q double-float (1) ((1 *))) 1)
     (dcopy (f2cl-lib:int-sub n 1) e 1

```

```

(f2cl-lib:array-slice q double-float ((+ n 1)) ((1 *))) 1)))
(cond
  (= iuplo 2)
  (setf qstart 5)
  (setf wstart (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
      (declare (ignore var-0 var-1))
      (setf cs var-2)
      (setf sn var-3)
      (setf r var-4)
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
      (setf
        (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
        (* sn (f2cl-lib:fref
          d-%data% ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)))
      (setf
        (f2cl-lib:fref d-%data%
          ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)
        (* cs (f2cl-lib:fref
          d-%data% ((f2cl-lib:int-add i 1)) ((1 *)) d-%offset%)))
      (cond
        (= icompq 1)
        (setf
          (f2cl-lib:fref q-%data%
            ((f2cl-lib:int-add i (f2cl-lib:int-mul 2 n)))
            ((1 *)) q-%offset%)
          cs)
        (setf
          (f2cl-lib:fref q-%data%
            ((f2cl-lib:int-add i (f2cl-lib:int-mul 3 n)))
            ((1 *)) q-%offset%)
          sn))
        (= icompq 2)
        (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%) cs)
        (setf
          (f2cl-lib:fref work-%data%
            ((f2cl-lib:int-add nm1 i)) ((1 *)) work-%offset%)
          (- sn))))))
  (cond
    (= icompq 0)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15)
      (dlasdq "U" 0 n 0 0 0 d e vt ldvt u ldu u ldu
        (f2cl-lib:array-slice work double-float (wstart) ((1 *))) info)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
           var-8 var-9 var-10 var-11 var-12 var-13 var-14))
(setf info var-15))
(go label40)))
(cond
  ((<= n smlsiz)
    (cond
      ((= icompq 2)
        (dlaset "A" n n zero one u ldu)
        (dlaset "A" n n zero one vt ldvt)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10 var-11 var-12 var-13 var-14 var-15)
          (dlasdq "U" 0 n n n 0 d e vt ldvt u ldu u ldu
                   (f2cl-lib:array-slice work double-float (wstart) ((1 *)))
                   info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12 var-13 var-14))
          (setf info var-15)))
      ((= icompq 1)
        (setf iu 1)
        (setf ivt (f2cl-lib:int-add iu n))
        (dlaset "A" n n zero one
                  (f2cl-lib:array-slice q double-float
                    ((+ iu
                       (f2cl-lib:int-mul
                        (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1)) n)))
                    ((1 *)))
                  n)
        (dlaset "A" n n zero one
                  (f2cl-lib:array-slice q double-float
                    ((+ ivt
                       (f2cl-lib:int-mul
                        (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1)) n)))
                    ((1 *)))
                  n)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10 var-11 var-12 var-13 var-14 var-15)
          (dlasdq "U" 0 n n n 0 d e
                   (f2cl-lib:array-slice q double-float
                    ((+ ivt
                       (f2cl-lib:int-mul
                        (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
                        n)))
                    ((1 *)))
                    n)
          (f2cl-lib:array-slice q double-float
            ((+ iu
               (f2cl-lib:int-mul

```

```

        (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
        n)))
    ((1 *)))
    n
    (f2cl-lib:array-slice q double-float
      ((+ iu
        (f2cl-lib:int-mul
          (f2cl-lib:int-add qstart (f2cl-lib:int-sub 1))
          n)))
      ((1 *)))
    n
    (f2cl-lib:array-slice work double-float (wstart) ((1 *)))
    info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9 var-10 var-11 var-12 var-13 var-14))
    (setf info var-15)))
    (go label40)))
(cond
  ((= icompq 2)
    (dlaset "A" n n zero one u ldu)
    (dlaset "A" n n zero one vt ldvt)))
(setf orgnrm (dlanst "M" n d e))
(if (= orgnrm zero) (go end_label))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 orgnrm one n 1 d n ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 orgnrm one nm1 1 e nm1 ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf ierr var-9))
(setf eps (dlamch "Epsilon"))
(setf mlvl
  (f2cl-lib:int-add
    (f2cl-lib:int
      (/
        (f2cl-lib:flog
          (/ (coerce (realpart n) 'double-float)
            (coerce (realpart (f2cl-lib:int-add smlsiz 1)) 'double-float)))
        (f2cl-lib:flog two)))
    1))
(setf smlszp (f2cl-lib:int-add smlsiz 1))
(cond
  ((= icompq 1)
    (setf iu 1)
    (setf ivt (f2cl-lib:int-add 1 smlsiz))

```

```

(setf difl (f2cl-lib:int-add ivt smlszp))
(setf difr (f2cl-lib:int-add difl mlvl))
(setf z (f2cl-lib:int-add difr (f2cl-lib:int-mul mlvl 2)))
(setf ic (f2cl-lib:int-add z mlvl))
(setf is (f2cl-lib:int-add ic 1))
(setf poles (f2cl-lib:int-add is 1))
(setf givnum (f2cl-lib:int-add poles (f2cl-lib:int-mul 2 mlvl)))
(setf k 1)
(setf givptr 2)
(setf perm 3)
(setf givcol (f2cl-lib:int-add perm mlvl)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i n) nil)
  (tagbody
    (cond
      ((< (abs (f2cl-lib:fref d (i) ((1 *)))) eps)
        (setf
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          (f2cl-lib:sign eps
            (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))))
      (setf start 1)
      (setf sqre 0)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i nm1) nil)
        (tagbody
          (cond
            ((or (< (abs (f2cl-lib:fref e (i) ((1 *)))) eps) (= i nm1))
              (cond
                ((< i nm1)
                  (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub i start) 1)))
                ((>= (abs (f2cl-lib:fref e (i) ((1 *)))) eps)
                  (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub n start) 1)))
                (t
                  (setf nsize (f2cl-lib:int-add (f2cl-lib:int-sub i start) 1)))
                (cond
                  ((= icompq 2)
                    (setf
                      (f2cl-lib:fref u-%data% (n n) ((1 ldu) (1 *)) u-%offset%)
                      (f2cl-lib:sign one
                        (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
                    (setf
                      (f2cl-lib:fref vt-%data% (n n) ((1 ldvt) (1 *)) vt-%offset%)
                      one))
                  ((= icompq 1)
                    (setf
                      (f2cl-lib:fref q-%data%
                        ((f2cl-lib:int-add n
                          (f2cl-lib:int-mul (f2cl-lib:int-sub qstart 1) n)))
                        ((1 *)) q-%offset%)
                      (f2cl-lib:sign one
```

```

      (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
(setf
  (f2cl-lib:fref q-%data%
    ((f2cl-lib:int-add n
      (f2cl-lib:int-mul
        (f2cl-lib:int-sub
          (f2cl-lib:int-add smlsiz qstart) 1) n)))
    ((1 *)) q-%offset%)
  one)))
(setf
  (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
  (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))))))
(cond
  ((= icompq 2)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11)
      (dlasd0 nsize sqre
        (f2cl-lib:array-slice d double-float (start) ((1 *)))
        (f2cl-lib:array-slice e double-float (start) ((1 *)))
        (f2cl-lib:array-slice u double-float
          (start start) ((1 ldu) (1 *)))
        ldu
        (f2cl-lib:array-slice vt double-float
          (start start) ((1 ldvt) (1 *)))
        ldvt
        smlsiz
        iwork
        (f2cl-lib:array-slice work double-float (wstart) ((1 *))) info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7 var-8 var-9 var-10))
      (setf info var-11)))
    (t
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9 var-10 var-11 var-12 var-13 var-14 var-15 var-16
         var-17 var-18 var-19 var-20 var-21 var-22 var-23)
        (dlasda icompq smlsiz nsize sqre
          (f2cl-lib:array-slice d double-float (start) ((1 *)))
          (f2cl-lib:array-slice e double-float (start) ((1 *)))
          (f2cl-lib:array-slice q double-float
            ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add iu qstart
              (f2cl-lib:int-sub 2)) n))) ((1 *)))
            n
            (f2cl-lib:array-slice q double-float
              ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add ivt qstart
                (f2cl-lib:int-sub 2)) n))) ((1 *)))
              (f2cl-lib:array-slice iq fixnum
                ((+ start (f2cl-lib:int-mul k n))) ((1 *)))
              (f2cl-lib:array-slice q double-float

```

```

      ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add difl qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add difr qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add z qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add poles qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice iq fixnum
  ((+ start (f2cl-lib:int-mul givptr n))) ((1 *)))
(f2cl-lib:array-slice iq fixnum
  ((+ start (f2cl-lib:int-mul givcol n))) ((1 *)))
n
(f2cl-lib:array-slice iq fixnum
  ((+ start (f2cl-lib:int-mul perm n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add givnum qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add ic qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice q double-float
  ((+ start (f2cl-lib:int-mul (f2cl-lib:int-add is qstart
                                   (f2cl-lib:int-sub 2)) n))) ((1 *)))
(f2cl-lib:array-slice work double-float (wstart) ((1 *)))
iwork
info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12
                  var-13 var-14 var-15 var-16 var-17 var-18
                  var-19 var-20 var-21 var-22))

  (setf info var-23))
(cond
  ((/= info 0) (go end_label))))
(setf start (f2cl-lib:int-add i 1))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 one orgnrm n 1 d n ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8))
  (setf ierr var-9))
label40
(f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
  (> ii n) nil)
(tagbody
  (setf i (f2cl-lib:int-sub ii 1))
  (setf kk i)
  (setf p (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))

```

```

(f2cl-lib:fdo (j ii (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (cond
    (> (f2cl-lib:fref d (j) ((1 *))) p)
    (setf kk j)
    (setf p (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
(cond
  (/= kk i)
  (setf (f2cl-lib:fref d-%data% (kk) ((1 *)) d-%offset%)
    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) p)
  (cond
    (= icompq 1)
    (setf (f2cl-lib:fref iq-%data% (i) ((1 *)) iq-%offset%) kk))
    (= icompq 2)
    (dswap n
      (f2cl-lib:array-slice u double-float (1 i) ((1 ldu) (1 *)))
      1
      (f2cl-lib:array-slice u double-float (1 kk) ((1 ldu) (1 *)))
      1)
    (dswap n
      (f2cl-lib:array-slice vt double-float (i 1) ((1 ldvt) (1 *)))
      ldvt
      (f2cl-lib:array-slice vt double-float (kk 1) ((1 ldvt) (1 *)))
      ldvt))))
  (= icompq 1)
  (setf (f2cl-lib:fref iq-%data% (i) ((1 *)) iq-%offset%) i))))))
(cond
  (= icompq 1)
  (cond
    (= iuplo 1)
    (setf (f2cl-lib:fref iq-%data% (n) ((1 *)) iq-%offset%) 1))
    (t
      (setf (f2cl-lib:fref iq-%data% (n) ((1 *)) iq-%offset%) 0))))))
(if (and (= iuplo 2) (= icompq 2))
  (dlasr "L" "V" "B" n n
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *))) u ldu))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dbdsqr LAPACK**— dbdsqr.input —**

```

)set break resume
)sys rm -f dbdsqr.output
)spool dbdsqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dbdsqr.help —

```

=====
dbdsqr examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DBDSQR - the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm

SYNOPSIS

```

SUBROUTINE DBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C,
                  LDC, WORK, INFO )

```

CHARACTER UPLO

INTEGER INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU

DOUBLE PRECISION C(LDC, *), D(*), E(*), U(LDU, *),
 VT(LDVT, *), WORK(*)

PURPOSE

DBDSQR computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

$$B = Q * S * P^{**T}$$

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns $U*Q$ instead of Q , and, if right singular vectors are requested, this subroutine returns $P^{**T}*VT$ instead of P^{**T} , for given real input matrices U and VT . When U and VT are the orthogonal matrices that reduce a general matrix A to bidiagonal form: $A = U*B*VT$, as computed by DGEBRD, then

$$A = (U*Q) * S * (P^{**T}*VT)$$

is the SVD of A . Optionally, the subroutine may also compute $Q^{**T}*C$ for a given real input matrix C .

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and

"Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

ARGUMENTS

- UPLO (input) CHARACTER*1
 = 'U': B is upper bidiagonal;
 = 'L': B is lower bidiagonal.
- N (input) INTEGER
 The order of the matrix B . $N \geq 0$.
- NCVT (input) INTEGER
 The number of columns of the matrix VT . $NCVT \geq 0$.
- NRU (input) INTEGER
 The number of rows of the matrix U . $NRU \geq 0$.
- NCC (input) INTEGER
 The number of columns of the matrix C . $NCC \geq 0$.
- D (input/output) DOUBLE PRECISION array, dimension (N)
 On entry, the n diagonal elements of the bidiagonal matrix B .
 On exit, if INFO=0, the singular values of B in decreasing order.
- E (input/output) DOUBLE PRECISION array, dimension (N-1)

On entry, the $N-1$ offdiagonal elements of the bidiagonal matrix B . On exit, if $\text{INFO} = 0$, E is destroyed; if $\text{INFO} > 0$, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT (input/output) DOUBLE PRECISION array, dimension $(\text{LDVT}, \text{NCVT})$
On entry, an N -by- NCVT matrix VT . On exit, VT is overwritten by $P^*T * VT$. Not referenced if $\text{NCVT} = 0$.

LDVT (input) INTEGER
The leading dimension of the array VT . $\text{LDVT} \geq \max(1, N)$ if $\text{NCVT} > 0$; $\text{LDVT} \geq 1$ if $\text{NCVT} = 0$.

U (input/output) DOUBLE PRECISION array, dimension (LDU, N)
On entry, an NRU -by- N matrix U . On exit, U is overwritten by $U * Q$. Not referenced if $\text{NRU} = 0$.

LDU (input) INTEGER
The leading dimension of the array U . $\text{LDU} \geq \max(1, \text{NRU})$.

C (input/output) DOUBLE PRECISION array, dimension (LDC, NCC)
On entry, an N -by- NCC matrix C . On exit, C is overwritten by $Q^*T * C$. Not referenced if $\text{NCC} = 0$.

LDC (input) INTEGER
The leading dimension of the array C . $\text{LDC} \geq \max(1, N)$ if $\text{NCC} > 0$; $\text{LDC} \geq 1$ if $\text{NCC} = 0$.

WORK (workspace) DOUBLE PRECISION array, dimension $(2*N)$
if $\text{NCVT} = \text{NRU} = \text{NCC} = 0$, $(\max(1, 4*N))$ otherwise

INFO (output) INTEGER
= 0: successful exit
< 0: If $\text{INFO} = -i$, the i -th argument had an illegal value
> 0: the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B ; if $\text{INFO} = i$, i elements of E have not converged to zero.

PARAMETERS

TOLMUL DOUBLE PRECISION, default = $\max(10, \min(100, \text{EPS}^{**(-1/8)}))$
TOLMUL controls the convergence criterion of the QR loop. If it is positive, $\text{TOLMUL} * \text{EPS}$ is the desired relative precision in the computed singular values. If it is negative, $\text{abs}(\text{TOLMUL} * \text{EPS} * \text{sigma_max})$ is the desired absolute accuracy in the computed singular values (corresponds to relative accuracy $\text{abs}(\text{TOLMUL} * \text{EPS})$ in the largest singular value. $\text{abs}(\text{TOLMUL})$ should be between 1 and $1/\text{EPS}$, and preferably between 10 (for fast convergence) and $.1/\text{EPS}$ (for there to be some accuracy in

the results). Default is to lose at either one eighth or 2 of the available decimal digits in each computed singular value (whichever is smaller).

MAXITR INTEGER, default = 6
 MAXITR controls the maximum number of passes of the algorithm through its inner loop. The algorithm stops (and so fails to converge) if the number of passes through the inner loop exceeds MAXITR*N**2.

— dbdsqr.f —

```

SUBROUTINE DBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U,
$                 LDU, C, LDC, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1999
*
*    .. Scalar Arguments ..
*    CHARACTER          UPLO
*    INTEGER            INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU
*
*    ..
*    .. Array Arguments ..
*    DOUBLE PRECISION   C( LDC, * ), D( * ), E( * ), U( LDU, * ),
$                      VT( LDVT, * ), WORK( * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
*    DOUBLE PRECISION   ZERO
*    PARAMETER          ( ZERO = 0.0D0 )
*    DOUBLE PRECISION   ONE
*    PARAMETER          ( ONE = 1.0D0 )
*    DOUBLE PRECISION   NEGONE
*    PARAMETER          ( NEGONE = -1.0D0 )
*    DOUBLE PRECISION   HNDRTH
*    PARAMETER          ( HNDRTH = 0.01D0 )
*    DOUBLE PRECISION   TEN
*    PARAMETER          ( TEN = 10.0D0 )
*    DOUBLE PRECISION   HNDRD
*    PARAMETER          ( HNDRD = 100.0D0 )
*    DOUBLE PRECISION   MEIGHTH
*    PARAMETER          ( MEIGHTH = -0.125D0 )

```

```

      INTEGER          MAXITR
      PARAMETER        ( MAXITR = 6 )
*
*   ..
*   .. Local Scalars ..
      LOGICAL          LOWER, ROTATE
      INTEGER          I, IDIR, ISUB, ITER, J, LL, LLL, M, MAXIT, NM1,
$                     NM12, NM13, OLDLL, OLDM
      DOUBLE PRECISION ABSE, ABSS, COSL, COSR, CS, EPS, F, G, H, MU,
$                     OLDCS, OLDSN, R, SHIFT, SIGMN, SIGMX, SINL,
$                     SINR, SLL, SMAX, SMIN, SMINL, SMINLO, SMINOA,
$                     SN, THRESH, TOL, TOLMUL, UNFL
*
*   ..
*   .. External Functions ..
      LOGICAL          LSAME
      DOUBLE PRECISION DLAMCH
      EXTERNAL         LSAME, DLAMCH
*
*   ..
*   .. External Subroutines ..
      EXTERNAL         DLARTG, DLAS2, DLASQ1, DLASR, DLASV2, DROT,
$                     DSCAL, DSWAP, XERBLA
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC        ABS, DBLE, MAX, MIN, SIGN, SQRT
*
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
      INFO = 0
      LOWER = LSAME( UPLO, 'L' )
      IF( .NOT.LSAME( UPLO, 'U' ) .AND. .NOT.LOWER ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( NCVT.LT.0 ) THEN
         INFO = -3
      ELSE IF( NRU.LT.0 ) THEN
         INFO = -4
      ELSE IF( NCC.LT.0 ) THEN
         INFO = -5
      ELSE IF( ( NCVT.EQ.0 .AND. LDVT.LT.1 ) .OR.
$             ( NCVT.GT.0 .AND. LDVT.LT.MAX( 1, N ) ) ) THEN
         INFO = -9
      ELSE IF( LDU.LT.MAX( 1, NRU ) ) THEN
         INFO = -11
      ELSE IF( ( NCC.EQ.0 .AND. LDC.LT.1 ) .OR.
$             ( NCC.GT.0 .AND. LDC.LT.MAX( 1, N ) ) ) THEN
         INFO = -13
      END IF
      IF( INFO.NE.0 ) THEN

```

```

        CALL XERBLA( 'DBDSQR', -INFO )
        RETURN
    END IF
    IF( N.EQ.0 )
$       RETURN
    IF( N.EQ.1 )
$       GO TO 160
*
*       ROTATE is true if any singular vectors desired, false otherwise
*
    ROTATE = ( NCVT.GT.0 ) .OR. ( NRU.GT.0 ) .OR. ( NCC.GT.0 )
*
*       If no singular vectors desired, use qd algorithm
*
    IF( .NOT.ROTATE ) THEN
        CALL DLASQ1( N, D, E, WORK, INFO )
        RETURN
    END IF
*
    NM1 = N - 1
    NM12 = NM1 + NM1
    NM13 = NM12 + NM1
    IDIR = 0
*
*       Get machine constants
*
    EPS = DLAMCH( 'Epsilon' )
    UNFL = DLAMCH( 'Safe minimum' )
*
*       If matrix lower bidiagonal, rotate to be upper bidiagonal
*       by applying Givens rotations on the left
*
    IF( LOWER ) THEN
        DO 10 I = 1, N - 1
            CALL DLARTG( D( I ), E( I ), CS, SN, R )
            D( I ) = R
            E( I ) = SN*D( I+1 )
            D( I+1 ) = CS*D( I+1 )
            WORK( I ) = CS
            WORK( NM1+I ) = SN
10      CONTINUE
*
*       Update singular vectors if desired
*
    IF( NRU.GT.0 )
$       CALL DLASR( 'R', 'V', 'F', NRU, N, WORK( 1 ), WORK( N ), U,
$               LDU )
    IF( NCC.GT.0 )
$       CALL DLASR( 'L', 'V', 'F', N, NCC, WORK( 1 ), WORK( N ), C,
$               LDC )

```

```

      END IF
*
*   Compute singular values to relative accuracy TOL
*   (By setting TOL to be negative, algorithm will compute
*   singular values to absolute accuracy ABS(TOL)*norm(input matrix))
*
      TOLMUL = MAX( TEN, MIN( HNRD, EPS**MEIGHTH ) )
      TOL = TOLMUL*EPS
*
*   Compute approximate maximum, minimum singular values
*
      SMAX = ZERO
      DO 20 I = 1, N
          SMAX = MAX( SMAX, ABS( D( I ) ) )
20  CONTINUE
      DO 30 I = 1, N - 1
          SMAX = MAX( SMAX, ABS( E( I ) ) )
30  CONTINUE
      SMINL = ZERO
      IF( TOL.GE.ZERO ) THEN
*
*       Relative accuracy desired
*
          SMINOA = ABS( D( 1 ) )
          IF( SMINOA.EQ.ZERO )
$             GO TO 50
          MU = SMINOA
          DO 40 I = 2, N
              MU = ABS( D( I ) )*( MU / ( MU+ABS( E( I-1 ) ) ) )
              SMINOA = MIN( SMINOA, MU )
              IF( SMINOA.EQ.ZERO )
$                  GO TO 50
40  CONTINUE
50  CONTINUE
          SMINOA = SMINOA / SQRT( DBLE( N ) )
          THRESH = MAX( TOL*SMINOA, MAXITR*N*N*UNFL )
      ELSE
*
*       Absolute accuracy desired
*
          THRESH = MAX( ABS( TOL )*SMAX, MAXITR*N*N*UNFL )
      END IF
*
*   Prepare for main iteration loop for the singular values
*   (MAXIT is the maximum number of passes through the inner
*   loop permitted before nonconvergence signalled.)
*
      MAXIT = MAXITR*N*N
      ITER = 0
      OLDLL = -1

```

```

        OLDM = -1
*
*   M points to last element of unconverged part of matrix
*
        M = N
*
*   Begin main iteration loop
*
60 CONTINUE
*
*   Check for convergence or exceeding iteration count
*
        IF( M.LE.1 )
$       GO TO 160
        IF( ITER.GT.MAXIT )
$       GO TO 200
*
*   Find diagonal block of matrix to work on
*
        IF( TOL.LT.ZERO .AND. ABS( D( M ) ).LE.THRESH )
$       D( M ) = ZERO
        SMAX = ABS( D( M ) )
        SMIN = SMAX
        DO 70 LLL = 1, M - 1
            LL = M - LLL
            ABSS = ABS( D( LL ) )
            ABSE = ABS( E( LL ) )
            IF( TOL.LT.ZERO .AND. ABSS.LE.THRESH )
$               D( LL ) = ZERO
            IF( ABSE.LE.THRESH )
$               GO TO 80
            SMIN = MIN( SMIN, ABSS )
            SMAX = MAX( SMAX, ABSS, ABSE )
70 CONTINUE
        LL = 0
        GO TO 90
80 CONTINUE
        E( LL ) = ZERO
*
*   Matrix splits since E(LL) = 0
*
        IF( LL.EQ.M-1 ) THEN
*
*       Convergence of bottom singular value, return to top of loop
*
            M = M - 1
            GO TO 60
        END IF
90 CONTINUE
        LL = LL + 1

```



```

*
*   E(LL) through E(M-1) are nonzero, E(LL-1) is zero
*
*   IF( LL.EQ.M-1 ) THEN
*
*       2 by 2 block, handle separately
*
*       CALL DLASV2( D( M-1 ), E( M-1 ), D( M ), SIGMN, SIGMX, SINR,
$           COSR, SINL, COSL )
*       D( M-1 ) = SIGMX
*       E( M-1 ) = ZERO
*       D( M ) = SIGMN
*
*       Compute singular vectors, if desired
*
*       IF( NCVT.GT.0 )
$           CALL DROT( NCVT, VT( M-1, 1 ), LDVT, VT( M, 1 ), LDVT, COSR,
$           SINR )
*       IF( NRU.GT.0 )
$           CALL DROT( NRU, U( 1, M-1 ), 1, U( 1, M ), 1, COSL, SINL )
*       IF( NCC.GT.0 )
$           CALL DROT( NCC, C( M-1, 1 ), LDC, C( M, 1 ), LDC, COSL,
$           SINL )
*       M = M - 2
*       GO TO 60
*   END IF
*
*   If working on new submatrix, choose shift direction
*   (from larger end diagonal element towards smaller)
*
*   IF( LL.GT.OLDM .OR. M.LT.OLDLL ) THEN
*       IF( ABS( D( LL ) ).GE.ABS( D( M ) ) ) THEN
*
*           Chase bulge from top (big end) to bottom (small end)
*
*           IDIR = 1
*       ELSE
*
*           Chase bulge from bottom (big end) to top (small end)
*
*           IDIR = 2
*       END IF
*   END IF
*
*   Apply convergence tests
*
*   IF( IDIR.EQ.1 ) THEN
*
*       Run convergence test in forward direction
*       First apply standard test to bottom of matrix

```

```

*
      IF( ABS( E( M-1 ) ).LE.ABS( TOL )*ABS( D( M ) ) .OR.
$      ( TOL.LT.ZERO .AND. ABS( E( M-1 ) ).LE.THRESH ) ) THEN
          E( M-1 ) = ZERO
          GO TO 60
      END IF
*
      IF( TOL.GE.ZERO ) THEN
*
*      If relative accuracy desired,
*      apply convergence criterion forward
*
          MU = ABS( D( LL ) )
          SMINL = MU
          DO 100 LLL = LL, M - 1
              IF( ABS( E( LLL ) ).LE.TOL*MU ) THEN
                  E( LLL ) = ZERO
                  GO TO 60
              END IF
              SMINL = MIN( SMINL, MU )
          100 CONTINUE
      END IF
*
      ELSE
*
*      Run convergence test in backward direction
*      First apply standard test to top of matrix
*
          IF( ABS( E( LL ) ).LE.ABS( TOL )*ABS( D( LL ) ) .OR.
$          ( TOL.LT.ZERO .AND. ABS( E( LL ) ).LE.THRESH ) ) THEN
              E( LL ) = ZERO
              GO TO 60
          END IF
*
          IF( TOL.GE.ZERO ) THEN
*
*      If relative accuracy desired,
*      apply convergence criterion backward
*
              MU = ABS( D( M ) )
              SMINL = MU
              DO 110 LLL = M - 1, LL, -1
                  IF( ABS( E( LLL ) ).LE.TOL*MU ) THEN
                      E( LLL ) = ZERO
                      GO TO 60
                  END IF
                  SMINL = MIN( SMINL, MU )
              110 CONTINUE
          END IF

```

```

                SMINL = MIN( SMINL, MU )
110          CONTINUE
            END IF
        END IF
        OLDLL = LL
        OLDM = M
*
*      Compute shift.  First, test if shifting would ruin relative
*      accuracy, and if so set the shift to zero.
*
        IF( TOL.GE.ZERO .AND. N*TOL*( SMINL / SMAX ).LE.
$          MAX( EPS, HNDRTH*TOL ) ) THEN
*
*          Use a zero shift to avoid loss of relative accuracy
*
            SHIFT = ZERO
        ELSE
*
*          Compute the shift from 2-by-2 block at end of matrix
*
            IF( IDIR.EQ.1 ) THEN
                SLL = ABS( D( LL ) )
                CALL DLAS2( D( M-1 ), E( M-1 ), D( M ), SHIFT, R )
            ELSE
                SLL = ABS( D( M ) )
                CALL DLAS2( D( LL ), E( LL ), D( LL+1 ), SHIFT, R )
            END IF
*
*          Test if shift negligible, and if so set to zero
*
            IF( SLL.GT.ZERO ) THEN
                IF( ( SHIFT / SLL )**2.LT.EPS )
$                  SHIFT = ZERO
            END IF
        END IF
*
*      Increment iteration count
*
        ITER = ITER + M - LL
*
*      If SHIFT = 0, do simplified QR iteration
*
        IF( SHIFT.EQ.ZERO ) THEN
            IF( IDIR.EQ.1 ) THEN
*
*              Chase bulge from top to bottom
*              Save cosines and sines for later singular vector updates
*
                CS = ONE
                OLDCS = ONE

```

```

DO 120 I = LL, M - 1
  CALL DLARTG( D( I )*CS, E( I ), CS, SN, R )
  IF( I.GT.LL )
    $      E( I-1 ) = OLDSN*R
    CALL DLARTG( OLDCS*R, D( I+1 )*SN, OLDCS, OLDSN, D( I ) )
    WORK( I-LL+1 ) = CS
    WORK( I-LL+1+NM1 ) = SN
    WORK( I-LL+1+NM12 ) = OLDCS
    WORK( I-LL+1+NM13 ) = OLDSN
120  CONTINUE
    H = D( M )*CS
    D( M ) = H*OLDCS
    E( M-1 ) = H*OLDSN
*
*      Update singular vectors
*
  IF( NCVT.GT.0 )
    $      CALL DLASR( 'L', 'V', 'F', M-LL+1, NCVT, WORK( 1 ),
    $              WORK( N ), VT( LL, 1 ), LDVT )
  IF( NRU.GT.0 )
    $      CALL DLASR( 'R', 'V', 'F', NRU, M-LL+1, WORK( NM12+1 ),
    $              WORK( NM13+1 ), U( 1, LL ), LDU )
  IF( NCC.GT.0 )
    $      CALL DLASR( 'L', 'V', 'F', M-LL+1, NCC, WORK( NM12+1 ),
    $              WORK( NM13+1 ), C( LL, 1 ), LDC )
*
*      Test convergence
*
  IF( ABS( E( M-1 ) ).LE.THRESH )
    $      E( M-1 ) = ZERO
*
  ELSE
*
*      Chase bulge from bottom to top
*      Save cosines and sines for later singular vector updates
*
    CS = ONE
    OLDCS = ONE
    DO 130 I = M, LL + 1, -1
      CALL DLARTG( D( I )*CS, E( I-1 ), CS, SN, R )
      IF( I.LT.M )
        $      E( I ) = OLDSN*R
        CALL DLARTG( OLDCS*R, D( I-1 )*SN, OLDCS, OLDSN, D( I ) )
        WORK( I-LL ) = CS
        WORK( I-LL+NM1 ) = -SN
        WORK( I-LL+NM12 ) = OLDCS
        WORK( I-LL+NM13 ) = -OLDSN
130  CONTINUE
    H = D( LL )*CS
    D( LL ) = H*OLDCS

```

```

      E( LL ) = H*OLDSN
*
*      Update singular vectors
*
      IF( NCVT.GT.0 )
$         CALL DLASR( 'L', 'V', 'B', M-LL+1, NCVT, WORK( NM12+1 ),
$                   WORK( NM13+1 ), VT( LL, 1 ), LDVT )
      IF( NRU.GT.0 )
$         CALL DLASR( 'R', 'V', 'B', NRU, M-LL+1, WORK( 1 ),
$                   WORK( N ), U( 1, LL ), LDU )
      IF( NCC.GT.0 )
$         CALL DLASR( 'L', 'V', 'B', M-LL+1, NCC, WORK( 1 ),
$                   WORK( N ), C( LL, 1 ), LDC )
*
*      Test convergence
*
      IF( ABS( E( LL ) ).LE.THRESH )
$         E( LL ) = ZERO
      END IF
      ELSE
*
*      Use nonzero shift
*
      IF( IDIR.EQ.1 ) THEN
*
*      Chase bulge from top to bottom
*      Save cosines and sines for later singular vector updates
*
      F = ( ABS( D( LL ) )-SHIFT )*
$         ( SIGN( ONE, D( LL ) )+SHIFT / D( LL ) )
      G = E( LL )
      DO 140 I = LL, M - 1
          CALL DLARTG( F, G, COSR, SINR, R )
          IF( I.GT.LL )
$             E( I-1 ) = R
          F = COSR*D( I ) + SINR*E( I )
          E( I ) = COSR*E( I ) - SINR*D( I )
          G = SINR*D( I+1 )
          D( I+1 ) = COSR*D( I+1 )
          CALL DLARTG( F, G, COSL, SINL, R )
          D( I ) = R
          F = COSL*E( I ) + SINL*D( I+1 )
          D( I+1 ) = COSL*D( I+1 ) - SINL*E( I )
          IF( I.LT.M-1 ) THEN
              G = SINL*E( I+1 )
              E( I+1 ) = COSL*E( I+1 )
          END IF
          WORK( I-LL+1 ) = COSR
          WORK( I-LL+1+NM1 ) = SINR
          WORK( I-LL+1+NM12 ) = COSL
      
```

```

      WORK( I-LL+1+NM13 ) = SINL
140      CONTINUE
      E( M-1 ) = F
*
*      Update singular vectors
*
      IF( NCVT.GT.0 )
$         CALL DLASR( 'L', 'V', 'F', M-LL+1, NCVT, WORK( 1 ),
$                   WORK( N ), VT( LL, 1 ), LDVT )
      IF( NRU.GT.0 )
$         CALL DLASR( 'R', 'V', 'F', NRU, M-LL+1, WORK( NM12+1 ),
$                   WORK( NM13+1 ), U( 1, LL ), LDU )
      IF( NCC.GT.0 )
$         CALL DLASR( 'L', 'V', 'F', M-LL+1, NCC, WORK( NM12+1 ),
$                   WORK( NM13+1 ), C( LL, 1 ), LDC )
*
*      Test convergence
*
      IF( ABS( E( M-1 ) ).LE.THRESH )
$         E( M-1 ) = ZERO
*
      ELSE
*
*      Chase bulge from bottom to top
*      Save cosines and sines for later singular vector updates
*
      F = ( ABS( D( M ) )-SHIFT )*( SIGN( ONE, D( M ) )+SHIFT /
$         D( M ) )
      G = E( M-1 )
      DO 150 I = M, LL + 1, -1
         CALL DLARTG( F, G, COSR, SINR, R )
         IF( I.LT.M )
$            E( I ) = R
         F = COSR*D( I ) + SINR*E( I-1 )
         E( I-1 ) = COSR*E( I-1 ) - SINR*D( I )
         G = SINR*D( I-1 )
         D( I-1 ) = COSR*D( I-1 )
         CALL DLARTG( F, G, COSL, SINL, R )
         D( I ) = R
         F = COSL*E( I-1 ) + SINL*D( I-1 )
         D( I-1 ) = COSL*D( I-1 ) - SINL*E( I-1 )
         IF( I.GT.LL+1 ) THEN
            G = SINL*E( I-2 )
            E( I-2 ) = COSL*E( I-2 )
         END IF
         WORK( I-LL ) = COSR
         WORK( I-LL+NM1 ) = -SINR
         WORK( I-LL+NM12 ) = COSL
         WORK( I-LL+NM13 ) = -SINL
150      CONTINUE

```

```

        E( LL ) = F
*
*      Test convergence
*
        IF( ABS( E( LL ) ).LE.THRESH )
$          E( LL ) = ZERO
*
*      Update singular vectors if desired
*
        IF( NCVT.GT.0 )
$          CALL DLASR( 'L', 'V', 'B', M-LL+1, NCVT, WORK( NM12+1 ),
$                    WORK( NM13+1 ), VT( LL, 1 ), LDVT )
        IF( NRU.GT.0 )
$          CALL DLASR( 'R', 'V', 'B', NRU, M-LL+1, WORK( 1 ),
$                    WORK( N ), U( 1, LL ), LDU )
        IF( NCC.GT.0 )
$          CALL DLASR( 'L', 'V', 'B', M-LL+1, NCC, WORK( 1 ),
$                    WORK( N ), C( LL, 1 ), LDC )
        END IF
    END IF
*
*      QR iteration finished, go back and check convergence
*
    GO TO 60
*
*      All singular values converged, so make them positive
*
160 CONTINUE
    DO 170 I = 1, N
        IF( D( I ).LT.ZERO ) THEN
            D( I ) = -D( I )
*
*          Change sign of singular vectors, if desired
*
            IF( NCVT.GT.0 )
$              CALL DSCAL( NCVT, NEGONE, VT( I, 1 ), LDVT )
            END IF
        END IF
    170 CONTINUE
*
*      Sort the singular values into decreasing order (insertion sort on
*      singular values, but only one transposition per singular vector)
*
    DO 190 I = 1, N - 1
*
*      Scan for smallest D(I)
*
        ISUB = 1
        SMIN = D( 1 )
        DO 180 J = 2, N + 1 - I
            IF( D( J ).LE.SMIN ) THEN

```

```

        ISUB = J
        SMIN = D( J )
        END IF
180    CONTINUE
        IF( ISUB.NE.N+1-I ) THEN
*
*          Swap singular values and vectors
*
        D( ISUB ) = D( N+1-I )
        D( N+1-I ) = SMIN
        IF( NCVT.GT.0 )
$          CALL DSWAP( NCVT, VT( ISUB, 1 ), LDVT, VT( N+1-I, 1 ),
$                      LDVT )
        IF( NRU.GT.0 )
$          CALL DSWAP( NRU, U( 1, ISUB ), 1, U( 1, N+1-I ), 1 )
        IF( NCC.GT.0 )
$          CALL DSWAP( NCC, C( ISUB, 1 ), LDC, C( N+1-I, 1 ), LDC )
        END IF
190    CONTINUE
        GO TO 220
*
*          Maximum number of iterations exceeded, failure to converge
*
200    CONTINUE
        INFO = 0
        DO 210 I = 1, N - 1
            IF( E( I ).NE.ZERO )
$              INFO = INFO + 1
210    CONTINUE
220    CONTINUE
        RETURN
*
*          End of DBDSQR
*
        END

```

— LAPACK dbdsqr —

```

(let* ((zero 0.0)
      (one 1.0)
      (negone (- 1.0))
      (hndrth 0.01)
      (ten 10.0)
      (hndrd 100.0)
      (meigth (- 0.125))
      (maxitr 6))

```



```

(declare (type (double-float 0.0 0.0) zero)
  (type (double-float 1.0 1.0) one)
  (type (double-float) negone)
  (type (double-float 0.01 0.01) hndrth)
  (type (double-float 10.0 10.0) ten)
  (type (double-float 100.0 100.0) hndrd)
  (type (double-float) meigth)
  (type (fixnum 6 6) maxitr))
(defun dbdsqr (uplo n ncvr nru ncc d e vt ldvt u ldu c ldc work info)
  (declare (type (simple-array double-float (*)) work c u vt e d)
    (type fixnum info ldc ldu ldvt ncc nru ncvr n)
    (type character uplo))
  (f2cl-lib:with-multi-array-data
    ((uplo character uplo-%data% uplo-%offset%)
     (d double-float d-%data% d-%offset%)
     (e double-float e-%data% e-%offset%)
     (vt double-float vt-%data% vt-%offset%)
     (u double-float u-%data% u-%offset%)
     (c double-float c-%data% c-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((abse 0.0) (abss 0.0) (cosl 0.0) (cosr 0.0) (cs 0.0) (eps 0.0)
      (f 0.0) (g 0.0) (h 0.0) (mu 0.0) (oldcs 0.0) (oldsn 0.0) (r 0.0)
      (shift 0.0) (sigmn 0.0) (sigmx 0.0) (sinl 0.0) (sinr 0.0)
      (sll 0.0) (smax 0.0) (smin 0.0) (sminl 0.0) (sminlo 0.0)
      (sminoa 0.0) (sn 0.0) (thresh 0.0) (tol 0.0) (tolmul 0.0)
      (unfl 0.0) (i 0) (idir 0) (isub 0) (iter 0) (j 0) (ll 0) (lll 0)
      (m 0) (maxit 0) (nm1 0) (nm12 0) (nm13 0) (oldl1 0) (oldm 0)
      (lower nil) (rotate nil))
      (declare (type (double-float) abse abss cosl cosr cs eps f g h mu oldcs
        oldsn r shift sigmn sigmx sinl sinr sll
        smax smin sminl sminlo sminoa sn thresh
        tol tolmul unfl)
        (type fixnum i idir isub iter j ll lll m maxit
          nm1 nm12 nm13 oldl1 oldm)
        (type (member t nil) lower rotate))
      (setf info 0)
      (setf lower (char-equal uplo #\L))
      (cond
        ((and (not (char-equal uplo #\U)) (not lower))
          (setf info -1))
        ((< n 0)
          (setf info -2))
        ((< ncvr 0)
          (setf info -3))
        ((< nru 0)
          (setf info -4))
        ((< ncc 0)
          (setf info -5))
        ((or (and (= ncvr 0) (< ldvt 1))
          (and (> ncvr 0)

```

```

        (< ldvt
         (max (the fixnum 1)
              (the fixnum n))))))
  (setf info -9))
  ((< ldu (max (the fixnum 1) (the fixnum nru)))
   (setf info -11))
  ((or (and (= ncc 0) (< ldc 1))
       (and (> ncc 0)
            (< ldc
             (max (the fixnum 1)
                  (the fixnum n))))))
   (setf info -13)))
(cond
 ((/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "DBDSQR" (f2cl-lib:int-sub info))
  (go end_label)))
(if (= n 0) (go end_label))
(if (= n 1) (go label160))
(setf rotate (or (> ncvr 0) (> nru 0) (> ncc 0)))
(cond
 (not rotate)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlasq1 n d e work info)
    (declare (ignore var-0 var-1 var-2 var-3))
    (setf info var-4))
  (go end_label)))
(setf nm1 (f2cl-lib:int-sub n 1))
(setf nm12 (f2cl-lib:int-add nm1 nm1))
(setf nm13 (f2cl-lib:int-add nm12 nm1))
(setf idir 0)
(setf eps (dlamch "Epsilon"))
(setf unfl (dlamch "Safe minimum"))
(cond
 (lower
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
     (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
              (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
     (declare (ignore var-0 var-1))
     (setf cs var-2)
     (setf sn var-3)
     (setf r var-4))
    (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
    (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
          (* sn
            (f2cl-lib:fref d-%data%

```

```

                                ((f2cl-lib:int-add i 1))
                                ((1 *))
                                d-%offset%)))
(setf (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add i 1))
                    ((1 *))
                    d-%offset%))
(* cs
  (f2cl-lib:fref d-%data%
                ((f2cl-lib:int-add i 1))
                ((1 *))
                d-%offset%)))
(setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%) cs)
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add nm1 i))
                    ((1 *))
                    work-%offset%))
      sn)))
(if (> nru 0)
  (dlasr "R" "V" "F" nru n
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *))) u ldu))
(if (> ncc 0)
  (dlasr "L" "V" "F" n ncc
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *))) c ldc))))
(setf tolmul (max ten (min hndrd (expt eps meigth))))
(setf tol (* tolmul eps))
(setf smax zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf smax
    (max smax
      (abs
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
(tagbody
  (setf smax
    (max smax
      (abs
        (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))))))
(setf sminl zero)
(cond
  ((>= tol zero)
    (tagbody
      (setf sminoa
        (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
      (if (= sminoa zero) (go label50))

```

```

(setf mu sminoa)
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
  ((> i n) nil)
  (tagbody
    (setf mu
      (*
        (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
        (/ mu
          (+ mu
            (abs
              (f2cl-lib:fref e-%data%
                ((f2cl-lib:int-sub i 1))
                ((1 *))
                e-%offset%)))))))
    (setf sminoa (min sminoa mu))
    (if (= sminoa zero) (go label150))))

label150
  (setf sminoa
    (/ sminoa (f2cl-lib:fsqrt (coerce (realpart n) 'double-float))))
  (setf thresh (max (* tol sminoa) (* maxitr n n unfl))))
  (t
    (setf thresh (max (* (abs tol) smax) (* maxitr n n unfl))))
  (setf maxit (f2cl-lib:int-mul maxitr n n))
  (setf iter 0)
  (setf oldll -1)
  (setf oldm -1)
  (setf m n)

label160
  (if (<= m 1) (go label160))
  (if (> iter maxit) (go label200))
  (if
    (and (< tol zero)
      (<= (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%))
        thresh))
    (setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) zero))
  (setf smax (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
  (setf smin smax)
  (f2cl-lib:fdo (lll 1 (f2cl-lib:int-add lll 1))
    ((> lll (f2cl-lib:int-add m (f2cl-lib:int-sub 1))) nil)
    (tagbody
      (setf ll (f2cl-lib:int-sub m lll))
      (setf abss (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%)))
      (setf abse (abs (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%)))
      (if (and (< tol zero) (<= abss thresh))
        (setf (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%) zero))
      (if (<= abse thresh) (go label180))
      (setf smin (min smin abss))
      (setf smax (max smax abss abse)))
    (setf ll 0)
    (go label190))

```

```

label80
  (setf (f2cl-lib:fref e-%data% (l1) ((1 *)) e-%offset%) zero)
  (cond
    ((= l1 (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
      (setf m (f2cl-lib:int-sub m 1))
      (go label60)))
label90
  (setf l1 (f2cl-lib:int-add l1 1))
  (cond
    ((= l1 (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
        (dlasv2
          (f2cl-lib:fref d-%data%
            ((f2cl-lib:int-sub m 1))
            ((1 *))
            d-%offset%)
          (f2cl-lib:fref e-%data%
            ((f2cl-lib:int-sub m 1))
            ((1 *))
            e-%offset%)
          (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) sigmn sigmx
          sinr cosr sinl cosl)
        (declare (ignore var-0 var-1 var-2))
        (setf sigmn var-3)
        (setf sigmx var-4)
        (setf sinr var-5)
        (setf cosr var-6)
        (setf sinl var-7)
        (setf cosl var-8))
        (setf (f2cl-lib:fref d-%data%
          ((f2cl-lib:int-sub m 1))
          ((1 *))
          d-%offset%)
          sigmx)
        (setf (f2cl-lib:fref e-%data%
          ((f2cl-lib:int-sub m 1))
          ((1 *))
          e-%offset%)
          zero)
        (setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) sigmn)
        (if (> ncvt 0)
          (drot ncvt
            (f2cl-lib:array-slice vt
              double-float
              ((+ m (f2cl-lib:int-sub 1)) 1)
              ((1 ldvt) (1 *)))
            ldvt
            (f2cl-lib:array-slice vt double-float (m 1) ((1 ldvt) (1 *)))
            ldvt cosr sinr)))

```

```

(if (> nru 0)
  (drot nru
    (f2cl-lib:array-slice u
      double-float
      (1 (f2cl-lib:int-sub m 1))
      ((1 ldu) (1 *)))
    1 (f2cl-lib:array-slice u double-float (1 m) ((1 ldu) (1 *))) 1
    cosl sinl))
(if (> ncc 0)
  (drot ncc
    (f2cl-lib:array-slice c
      double-float
      ((+ m (f2cl-lib:int-sub 1)) 1)
      ((1 ldc) (1 *)))
    ldc (f2cl-lib:array-slice c double-float (m 1) ((1 ldc) (1 *)))
    ldc cosl sinl))
(setf m (f2cl-lib:int-sub m 2))
(go label60)))
(cond
  ((or (> ll oldm) (< m oldl1))
    (cond
      ((>= (abs (f2cl-lib:fref d (ll) ((1 *))))
        (abs (f2cl-lib:fref d (m) ((1 *)))))
        (setf idir 1))
      (t
        (setf idir 2))))))
(cond
  ((= idir 1)
    (cond
      ((or
        (<=
          (abs
            (f2cl-lib:fref e
              ((f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
              ((1 *))))
          (* (abs tol) (abs (f2cl-lib:fref d (m) ((1 *))))))
        (and (< tol zero)
          (<=
            (abs
              (f2cl-lib:fref e
                ((f2cl-lib:int-add m
                  (f2cl-lib:int-sub 1)))
                ((1 *))))
            thresh))))
      (setf (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub m 1))
        ((1 *))
        e-%offset%)
        zero)
      (go label60)))

```

```

(cond
  ((>= tol zero)
    (setf mu (abs (f2cl-lib:fref d-%data% (l1) ((1 *)) d-%offset%)))
    (setf sminl mu)
    (f2cl-lib:fdo (l1l l1 (f2cl-lib:int-add l1l 1))
      ((> l1l (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
        nil)
      (tagbody
        (cond
          ((<= (abs (f2cl-lib:fref e (l1l) ((1 *)))) (* tol mu))
            (setf (f2cl-lib:fref e-%data% (l1l) ((1 *)) e-%offset%)
              zero)
            (go label60)))
          (setf sminlo sminl)
          (setf mu
            (*
              (abs
                (f2cl-lib:fref d-%data%
                  ((f2cl-lib:int-add l1l 1))
                  ((1 *))
                  d-%offset%))
              (/ mu
                (+ mu
                  (abs
                    (f2cl-lib:fref e-%data%
                      (l1l)
                      ((1 *))
                      e-%offset%)))))))
            (setf sminl (min sminl mu)))))))
  (t
    (cond
      ((or
        (<= (abs (f2cl-lib:fref e (l1) ((1 *))))
          (* (abs tol) (abs (f2cl-lib:fref d (l1) ((1 *))))))
        (and (< tol zero)
          (<= (abs (f2cl-lib:fref e (l1) ((1 *)))) thresh)))
        (setf (f2cl-lib:fref e-%data% (l1) ((1 *)) e-%offset%) zero)
        (go label60)))
      (cond
        ((>= tol zero)
          (setf mu (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
          (setf sminl mu)
          (f2cl-lib:fdo (l1l (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
            (f2cl-lib:int-add l1l (f2cl-lib:int-sub 1)))
            ((> l1l l1) nil)
            (tagbody
              (cond
                ((<= (abs (f2cl-lib:fref e (l1l) ((1 *)))) (* tol mu))
                  (setf (f2cl-lib:fref e-%data% (l1l) ((1 *)) e-%offset%)
                    zero)

```

```

        (go label60)))
      (setf sminlo sminl)
      (setf mu
        (*
          (abs
            (f2cl-lib:fref d-%data% (l1l) ((1 *)) d-%offset%))
          (/ mu
            (+ mu
              (abs
                (f2cl-lib:fref e-%data%
                              (l1l)
                              ((1 *))
                              e-%offset%))))))
        (setf sminl (min sminl mu))))))
    (setf oldl1 l1)
    (setf oldm m)
    (cond
      ((and (>= tol zero)
            (<= (* n tol (f2cl-lib:f2cl/ sminl smax))
              (max eps (* hndrth tol)))))
      (setf shift zero))
    (t
      (cond
        ((= idir 1)
          (setf s1l (abs (f2cl-lib:fref d-%data% (l1) ((1 *)) d-%offset%)))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
            (dlas2
              (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-sub m 1))
                            ((1 *))
                            d-%offset%)
              (f2cl-lib:fref e-%data%
                            ((f2cl-lib:int-sub m 1))
                            ((1 *))
                            e-%offset%)
              (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) shift r)
            (declare (ignore var-0 var-1 var-2))
            (setf shift var-3)
            (setf r var-4)))
          (t
            (setf s1l (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%)))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlas2 (f2cl-lib:fref d-%data% (l1) ((1 *)) d-%offset%)
                    (f2cl-lib:fref e-%data% (l1) ((1 *)) e-%offset%)
                    (f2cl-lib:fref d-%data%
                                  ((f2cl-lib:int-add l1 1))
                                  ((1 *))
                                  d-%offset%)
                    shift r)
              (declare (ignore var-0 var-1 var-2))

```



```

      (setf shift var-3)
      (setf r var-4)))
    (cond
      ((> sll zero)
        (if (< (expt (/ shift sll) 2) eps) (setf shift zero))))))
    (setf iter (f2cl-lib:int-sub (f2cl-lib:int-add iter m) 1))
    (cond
      ((= shift zero)
        (cond
          ((= idir 1)
            (setf cs one)
            (setf oldcs one)
            (f2cl-lib:fdo (i 1) (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                nil)
            (tagbody
              (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlartg
                  (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) cs)
                  (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
                (declare (ignore var-0 var-1))
                (setf cs var-2)
                (setf sn var-3)
                (setf r var-4))
              (if (> i 1)
                (setf (f2cl-lib:fref e-%data%
                                      ((f2cl-lib:int-sub i 1))
                                      ((1 *))
                                      e-%offset%)
                      (* oldsn r)))
                (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                  (dlartg (* oldcs r)
                    (*
                      (f2cl-lib:fref d-%data%
                                      ((f2cl-lib:int-add i 1))
                                      ((1 *))
                                      d-%offset%)
                      sn)
                    oldcs oldsn
                    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
                  (declare (ignore var-0 var-1))
                  (setf oldcs var-2)
                  (setf oldsn var-3)
                  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%
                                      var-4))
                    (setf (f2cl-lib:fref work-%data%
                                      ((f2cl-lib:int-add
                                        (f2cl-lib:int-sub i 1)
                                        1))
                                      ((1 *))

```

```

                                work-%offset%)
      cs)
    (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add
                          (f2cl-lib:int-sub i ll)
                          1
                          nm1))
                        ((1 *)))
          work-%offset%)

      sn)
    (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add
                          (f2cl-lib:int-sub i ll)
                          1
                          nm12))
                        ((1 *)))
          work-%offset%)

      oldcs)
    (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add
                          (f2cl-lib:int-sub i ll)
                          1
                          nm13))
                        ((1 *)))
          work-%offset%)

      oldsn)))
(setf h (* (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) cs))
(setf (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%) (* h oldcs))
(setf (f2cl-lib:fref e-%data%
                    ((f2cl-lib:int-sub m 1))
                    ((1 *))
                    e-%offset%)
      (* h oldsn))
(if (> ncv 0)
  (dlsr "L" "V" "F"
        (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncv
        (f2cl-lib:array-slice work double-float (1) ((1 *)))
        (f2cl-lib:array-slice work double-float (n) ((1 *)))
        (f2cl-lib:array-slice vt
                               double-float
                               (ll 1)
                               ((1 ldvt) (1 *))))
  ldvt))
(if (> nru 0)
  (dlsr "R" "V" "F" nru
        (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
        (f2cl-lib:array-slice work
                              double-float
                              ((+ nm12 1))
                              ((1 *)))
```

```

(f2cl-lib:array-slice work
  double-float
  ((+ nm13 1))
  ((1 *)))
(f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
ldu))
(if (> ncc 0)
  (dlasr "L" "V" "F"
    (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
    (f2cl-lib:array-slice work
      double-float
      ((+ nm12 1))
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      ((+ nm13 1))
      ((1 *)))
    (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
    ldc))
(if
  (<=
    (abs
      (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub m 1))
        ((1 *))
        e-%offset%))
    thresh)
  (setf (f2cl-lib:fref e-%data%
    ((f2cl-lib:int-sub m 1))
    ((1 *))
    e-%offset%)
    zero)))
(t
  (setf cs one)
  (setf oldcs one)
  (f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    ((> i (f2cl-lib:int-add ll 1)) nil)
    (tagbody
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
        (dlartg
          (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) cs)
          (f2cl-lib:fref e-%data%
            ((f2cl-lib:int-sub i 1))
            ((1 *))
            e-%offset%)
          cs sn r)
        (declare (ignore var-0 var-1))
        (setf cs var-2)
        (setf sn var-3)
        (setf r var-4))

```

```

(if (< i m)
  (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
        (* oldsn r)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg (* oldcs r)
    (*
      (f2cl-lib:fref d-%data%
        ((f2cl-lib:int-sub i 1))
        ((1 *))
        d-%offset%)

      sn)
    oldcs oldsn
    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (declare (ignore var-0 var-1))
  (setf oldcs var-2)
  (setf oldsn var-3)
  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        var-4))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-sub i 11))
  ((1 *))
  work-%offset%)

  cs)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    nm1))
  ((1 *))
  work-%offset%)

  (- sn))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    nm12))
  ((1 *))
  work-%offset%)

  oldcs)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    nm13))
  ((1 *))
  work-%offset%)

  (- oldsn))))
(setf h (* (f2cl-lib:fref d-%data% (11) ((1 *)) d-%offset%) cs))
(setf (f2cl-lib:fref d-%data% (11) ((1 *)) d-%offset%)
      (* h oldcs))
(setf (f2cl-lib:fref e-%data% (11) ((1 *)) e-%offset%)
      (* h oldsn))
(if (> ncvt 0)

```

```

(dlasr "L" "V" "B"
  (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncv
  (f2cl-lib:array-slice work
    double-float
    ((+ nm12 1))
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    ((+ nm13 1))
    ((1 *)))
  (f2cl-lib:array-slice vt
    double-float
    (ll 1)
    ((1 ldvt) (1 *)))
  ldvt))
(if (> nru 0)
  (dlasr "R" "V" "B" nru
    (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *)))
    (f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
    ldu))
(if (> ncc 0)
  (dlasr "L" "V" "B"
    (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *)))
    (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
    ldc))
(if
  (<= (abs (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%))
    thresh)
  (setf (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%) zero))))
(t
  (cond
    ((= idir 1)
      (setf f
        (*
          (-
            (abs (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%))
            shift)
          (+
            (f2cl-lib:sign one
              (f2cl-lib:fref d-%data%
                (ll)
                ((1 *))
                d-%offset%))
            (/ shift
              (f2cl-lib:fref d-%data% (ll) ((1 *)) d-%offset%))))))
      (setf g (f2cl-lib:fref e-%data% (ll) ((1 *)) e-%offset%))

```

```

(f2cl-lib:fdo (i ll (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
   nil)
(tagbody
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg f g cosr sinr r)
    (declare (ignore var-0 var-1))
    (setf cosr var-2)
    (setf sinr var-3)
    (setf r var-4))
  (if (> i ll)
    (setf (f2cl-lib:fref e-%data%
                        ((f2cl-lib:int-sub i 1))
                        ((1 *))
                        e-%offset%)
          r))
  (setf f
    (+
      (* cosr
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
      (* sinr
        (f2cl-lib:fref e-%data%
                        (i)
                        ((1 *))
                        e-%offset%))))
  (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
    (-
      (* cosr
        (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))
      (* sinr
        (f2cl-lib:fref d-%data%
                        (i)
                        ((1 *))
                        d-%offset%))))
  (setf g
    (* sinr
      (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add i 1))
                    ((1 *))
                    d-%offset%)))
  (setf (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add i 1))
                    ((1 *))
                    d-%offset%)
    (* cosr
      (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add i 1))
                    ((1 *))
                    d-%offset%)))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)

```

```

      (dlartg f g cosl sinl r)
      (declare (ignore var-0 var-1))
      (setf cosl var-2)
      (setf sinl var-3)
      (setf r var-4))
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
      (setf f
        (+
          (* cosl
            (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))
          (* sinl
            (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%))))))
      (setf (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%)
        (-
          (* cosl
            (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          d-%offset%))
          (* sinl
            (f2cl-lib:fref e-%data%
                          (i)
                          ((1 *))
                          e-%offset%))))))
      (cond
        ((< i (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
          (setf g
            (* sinl
              (f2cl-lib:fref e-%data%
                            ((f2cl-lib:int-add i 1))
                            ((1 *))
                            e-%offset%)))
            (setf (f2cl-lib:fref e-%data%
                              ((f2cl-lib:int-add i 1))
                              ((1 *))
                              e-%offset%)
              (* cosl
                (f2cl-lib:fref e-%data%
                              ((f2cl-lib:int-add i 1))
                              ((1 *))
                              e-%offset%))))))
      (setf (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add
                            (f2cl-lib:int-sub i 11)

```

```

1))
((1 *))
work-%offset%)

cosr)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    1
    nm1))
  ((1 *))
  work-%offset%)

sinr)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    1
    nm12))
  ((1 *))
  work-%offset%)

cosl)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub i 11)
    1
    nm13))
  ((1 *))
  work-%offset%)

sinl)))
(setf (f2cl-lib:fref e-%data%
  ((f2cl-lib:int-sub m 1))
  ((1 *))
  e-%offset%)

f)
(if (> ncv 0)
  (dlsr "L" "V" "F"
    (f2cl-lib:int-add (f2cl-lib:int-sub m 11) 1) ncv
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (n) ((1 *)))
    (f2cl-lib:array-slice vt
      double-float
      (11 1)
      ((1 ldvt) (1 *)))
    ldvt))
(if (> nru 0)
  (dlsr "R" "V" "F" nru
    (f2cl-lib:int-add (f2cl-lib:int-sub m 11) 1)
    (f2cl-lib:array-slice work
      double-float
      ((+ nm12 1))
      ((1 *)))

```



```

(f2cl-lib:array-slice work
  double-float
  ((+ nm13 1))
  ((1 *)))
(f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
ldu))
(if (> ncc 0)
  (dlasr "L" "V" "F"
    (f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
    (f2cl-lib:array-slice work
      double-float
      ((+ nm12 1))
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      ((+ nm13 1))
      ((1 *)))
    (f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
    ldc))
(if
  (<=
    (abs
      (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub m 1))
        ((1 *))
        e-%offset%))
    thresh)
  (setf (f2cl-lib:fref e-%data%
    ((f2cl-lib:int-sub m 1))
    ((1 *))
    e-%offset%)
    zero)))
(t
  (setf f
    (*
      (- (abs (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%))
        shift)
      (+
        (f2cl-lib:sign one
          (f2cl-lib:fref d-%data%
            (m)
            ((1 *))
            d-%offset%))
        (/ shift
          (f2cl-lib:fref d-%data% (m) ((1 *)) d-%offset%))))))
  (setf g
    (f2cl-lib:fref e-%data%
      ((f2cl-lib:int-sub m 1))
      ((1 *))
      e-%offset%))

```

```

(f2cl-lib:fdo (i m (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
  (> i (f2cl-lib:int-add 11 1)) nil)
(tagbody
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlartg f g cosr sinr r)
    (declare (ignore var-0 var-1))
    (setf cosr var-2)
    (setf sinr var-3)
    (setf r var-4))
  (if (< i m)
    (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) r))
  (setf f
    (+
      (* cosr
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
      (* sinr
        (f2cl-lib:fref e-%data%
          ((f2cl-lib:int-sub i 1))
          ((1 *))
          e-%offset%))))
    (setf (f2cl-lib:fref e-%data%
      ((f2cl-lib:int-sub i 1))
      ((1 *))
      e-%offset%)
      (-
        (* cosr
          (f2cl-lib:fref e-%data%
            ((f2cl-lib:int-sub i 1))
            ((1 *))
            e-%offset%))
        (* sinr
          (f2cl-lib:fref d-%data%
            (i)
            ((1 *))
            d-%offset%))))
    (setf g
      (* sinr
        (f2cl-lib:fref d-%data%
          ((f2cl-lib:int-sub i 1))
          ((1 *))
          d-%offset%)))
    (setf (f2cl-lib:fref d-%data%
      ((f2cl-lib:int-sub i 1))
      ((1 *))
      d-%offset%)
      (* cosr
        (f2cl-lib:fref d-%data%
          ((f2cl-lib:int-sub i 1))
          ((1 *))
          d-%offset%)))

```

```

(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg f g cosl sinl r)
  (declare (ignore var-0 var-1))
  (setf cosl var-2)
  (setf sinl var-3)
  (setf r var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
(setf f
  (+
    (* cosl
      (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub i 1))
        ((1 *))
        e-%offset%))
    (* sinl
      (f2cl-lib:fref d-%data%
        ((f2cl-lib:int-sub i 1))
        ((1 *))
        d-%offset%))))))
(setf (f2cl-lib:fref d-%data%
  ((f2cl-lib:int-sub i 1))
  ((1 *))
  d-%offset%)
  (-
    (* cosl
      (f2cl-lib:fref d-%data%
        ((f2cl-lib:int-sub i 1))
        ((1 *))
        d-%offset%))
    (* sinl
      (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub i 1))
        ((1 *))
        e-%offset%))))))
(cond
  ((> i (f2cl-lib:int-add ll 1))
    (setf g
      (* sinl
        (f2cl-lib:fref e-%data%
          ((f2cl-lib:int-sub i 2))
          ((1 *))
          e-%offset%))
      (setf (f2cl-lib:fref e-%data%
        ((f2cl-lib:int-sub i 2))
        ((1 *))
        e-%offset%)
        (* cosl
          (f2cl-lib:fref e-%data%
            ((f2cl-lib:int-sub i 2))
            ((1 *))
            e-%offset%)
          (f2cl-lib:fref d-%data%
            ((f2cl-lib:int-sub i 2))
            ((1 *))
            d-%offset%))))))

```

```

                                e-%offset%))))))
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-sub i 11))
                    ((1 *))
                    work-%offset%))

    cosr)
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add
                     (f2cl-lib:int-sub i 11)
                     nm1))
                    ((1 *))
                    work-%offset%))

    (- sinr))
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add
                     (f2cl-lib:int-sub i 11)
                     nm12))
                    ((1 *))
                    work-%offset%))

    cosl)
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add
                     (f2cl-lib:int-sub i 11)
                     nm13))
                    ((1 *))
                    work-%offset%))

    (- sinl))))
(setf (f2cl-lib:fref e-%data% (11) ((1 *)) e-%offset%) f)
(if
  (<= (abs (f2cl-lib:fref e-%data% (11) ((1 *)) e-%offset%))
       thresh)
  (setf (f2cl-lib:fref e-%data% (11) ((1 *)) e-%offset%) zero))
(if (> ncv 0)
  (dlsr "L" "V" "B"
    (f2cl-lib:int-add (f2cl-lib:int-sub m 11) 1) ncv
    (f2cl-lib:array-slice work
      double-float
      ((+ nm12 1))
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      ((+ nm13 1))
      ((1 *)))
    (f2cl-lib:array-slice vt
      double-float
      (11 1)
      ((1 ldvt) (1 *)))
    ldvt))
(if (> nru 0)
  (dlsr "R" "V" "B" nru

```

```

(f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1)
(f2cl-lib:array-slice work double-float (1) ((1 *)))
(f2cl-lib:array-slice work double-float (n) ((1 *)))
(f2cl-lib:array-slice u double-float (1 ll) ((1 ldu) (1 *)))
ldu))
(if (> ncc 0)
(dlasr "L" "V" "B"
(f2cl-lib:int-add (f2cl-lib:int-sub m ll) 1) ncc
(f2cl-lib:array-slice work double-float (1) ((1 *)))
(f2cl-lib:array-slice work double-float (n) ((1 *)))
(f2cl-lib:array-slice c double-float (ll 1) ((1 ldc) (1 *)))
ldc))))))
(go label60)
label160
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i n) nil)
(tagbody
(cond
(< (f2cl-lib:fref d (i) ((1 *))) zero)
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
(- (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))
(if (> ncv1 0)
(dscal ncv1 negone
(f2cl-lib:array-slice vt
double-float
(i 1)
((1 ldvt) (1 *)))
ldvt))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
(tagbody
(setf isub 1)
(setf smin (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
(> j (f2cl-lib:int-add n 1 (f2cl-lib:int-sub i)))
nil)
(tagbody
(cond
(<= (f2cl-lib:fref d (j) ((1 *))) smin)
(setf isub j)
(setf smin
(f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
(cond
(/= isub (f2cl-lib:int-add n 1 (f2cl-lib:int-sub i)))
(setf (f2cl-lib:fref d-%data% (isub) ((1 *)) d-%offset%)
(f2cl-lib:fref d-%data%
((f2cl-lib:int-sub (f2cl-lib:int-add n 1)
i))
((1 *))
d-%offset%))

```

```

(setf (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-add n 1)
                                         i))
                    ((1 *))
                    d-%offset%)
      smin)
(if (> ncv 0)
    (dswap ncv
      (f2cl-lib:array-slice vt
                            double-float
                            (isub 1)
                            ((1 ldvt) (1 *)))
      ldvt
      (f2cl-lib:array-slice vt
                            double-float
                            ((+ n 1 (f2cl-lib:int-sub i)) 1)
                            ((1 ldvt) (1 *)))
      ldvt))
(if (> nru 0)
    (dswap nru
      (f2cl-lib:array-slice u
                            double-float
                            (1 isub)
                            ((1 ldu) (1 *)))
      1
      (f2cl-lib:array-slice u
                            double-float
                            (1
                              (f2cl-lib:int-sub
                               (f2cl-lib:int-add n 1)
                               i))
                            ((1 ldu) (1 *)))
      1))
(if (> ncc 0)
    (dswap ncc
      (f2cl-lib:array-slice c
                            double-float
                            (isub 1)
                            ((1 ldc) (1 *)))
      ldc
      (f2cl-lib:array-slice c
                            double-float
                            ((+ n 1 (f2cl-lib:int-sub i)) 1)
                            ((1 ldc) (1 *)))
      ldc))))))
(go end_label)
label200
(setf info 0)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)

```

```

      (tagbody
        (if (/= (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) zero)
          (setf info (f2cl-lib:int-add info 1))))
end_label
(return
 (values nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         info))))

```

—————

ddisna LAPACK

— ddisna.input —

```

)set break resume
)sys rm -f ddisna.output
)spool ddisna.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

—————

— ddisna.help —

```

=====
ddisna examples
=====

```

```
=====
Man Page Details
=====
```

NAME

DDISNA - the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix

SYNOPSIS

```
SUBROUTINE DDISNA( JOB, M, N, D, SEP, INFO )
```

```

      CHARACTER      JOB
      INTEGER        INFO, M, N
      DOUBLE         PRECISION D( * ), SEP( * )
```

PURPOSE

DDISNA computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix. The reciprocal condition number is the 'gap' between the corresponding eigenvalue or singular value and the nearest other one.

The bound on the error, measured by angle in radians, in the I-th computed vector is given by

$$\text{DLAMCH}('E') * (\text{ANORM} / \text{SEP}(I))$$

where $\text{ANORM} = 2\text{-norm}(A) = \max(\text{abs}(D(j)))$. $\text{SEP}(I)$ is not allowed to be smaller than $\text{DLAMCH}('E') * \text{ANORM}$ in order to limit the size of the error bound.

DDISNA may also be used to compute error bounds for eigenvectors of the generalized symmetric definite eigenproblem.

ARGUMENTS

```

      JOB      (input) CHARACTER*1
               Specifies for which problem the reciprocal condition numbers
               should be computed:
               = 'E': the eigenvectors of a symmetric/Hermitian matrix;
               = 'L': the left singular vectors of a general matrix;
               = 'R': the right singular vectors of a general matrix.

      M        (input) INTEGER
               The number of rows of the matrix. M >= 0.

      N        (input) INTEGER
```


If JOB = 'L' or 'R', the number of columns of the matrix, in which case N >= 0. Ignored if JOB = 'E'.

D (input) DOUBLE PRECISION array, dimension (M) if JOB = 'E' dimension (min(M,N)) if JOB = 'L' or 'R' The eigenvalues (if JOB = 'E') or singular values (if JOB = 'L' or 'R') of the matrix, in either increasing or decreasing order. If singular values, they must be non-negative.

SEP (output) DOUBLE PRECISION array, dimension (M) if JOB = 'E' dimension (min(M,N)) if JOB = 'L' or 'R' The reciprocal condition numbers of the vectors.

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.

— ddisna.f —

```

SUBROUTINE DDISNA( JOB, M, N, D, SEP, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  September 30, 1994
*
*  .. Scalar Arguments ..
CHARACTER          JOB
INTEGER            INFO, M, N
*
*  ..
*
*  .. Array Arguments ..
DOUBLE PRECISION   D( * ), SEP( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION   ZERO
PARAMETER          ( ZERO = 0.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
LOGICAL            DECR, EIGEN, INCR, LEFT, RIGHT, SING
INTEGER            I, K
DOUBLE PRECISION   ANORM, EPS, NEWGAP, OLDGAP, SAFMIN, THRESH
*
*  ..
*
*  .. External Functions ..

```

```

LOGICAL          LSAME
DOUBLE PRECISION DLAMCH
EXTERNAL         LSAME, DLAMCH
*
* ..
* .. Intrinsic Functions ..
INTRINSIC         ABS, MAX, MIN
*
* ..
* .. External Subroutines ..
EXTERNAL         XERBLA
*
* ..
* .. Executable Statements ..
*
* Test the input arguments
*
INFO = 0
EIGEN = LSAME( JOB, 'E' )
LEFT = LSAME( JOB, 'L' )
RIGHT = LSAME( JOB, 'R' )
SING = LEFT .OR. RIGHT
IF( EIGEN ) THEN
    K = M
ELSE IF( SING ) THEN
    K = MIN( M, N )
END IF
IF( .NOT.EIGEN .AND. .NOT.SING ) THEN
    INFO = -1
ELSE IF( M.LT.0 ) THEN
    INFO = -2
ELSE IF( K.LT.0 ) THEN
    INFO = -3
ELSE
    INCR = .TRUE.
    DECR = .TRUE.
    DO 10 I = 1, K - 1
        IF( INCR )
$           INCR = INCR .AND. D( I ).LE.D( I+1 )
        IF( DECR )
$           DECR = DECR .AND. D( I ).GE.D( I+1 )
10    CONTINUE
    IF( SING .AND. K.GT.0 ) THEN
        IF( INCR )
$           INCR = INCR .AND. ZERO.LE.D( 1 )
        IF( DECR )
$           DECR = DECR .AND. D( K ).GE.ZERO
    END IF
    IF( .NOT.( INCR .OR. DECR ) )
$       INFO = -4
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DDISNA', -INFO )

```

```

        RETURN
    END IF

*
*   Quick return if possible
*
    IF( K.EQ.0 )
$       RETURN
*
*   Compute reciprocal condition numbers
*
    IF( K.EQ.1 ) THEN
        SEP( 1 ) = DLAMCH( 'O' )
    ELSE
        OLDGAP = ABS( D( 2 )-D( 1 ) )
        SEP( 1 ) = OLDGAP
        DO 20 I = 2, K - 1
            NEWGAP = ABS( D( I+1 )-D( I ) )
            SEP( I ) = MIN( OLDGAP, NEWGAP )
            OLDGAP = NEWGAP
20      CONTINUE
        SEP( K ) = OLDGAP
    END IF
    IF( SING ) THEN
        IF( ( LEFT .AND. M.GT.N ) .OR. ( RIGHT .AND. M.LT.N ) ) THEN
            IF( INCR )
$               SEP( 1 ) = MIN( SEP( 1 ), D( 1 ) )
            IF( DECR )
$               SEP( K ) = MIN( SEP( K ), D( K ) )
        END IF
    END IF

*
*   Ensure that reciprocal condition numbers are not less than
*   threshold, in order to limit the size of the error bound
*
    EPS = DLAMCH( 'E' )
    SAFMIN = DLAMCH( 'S' )
    ANORM = MAX( ABS( D( 1 ) ), ABS( D( K ) ) )
    IF( ANORM.EQ.ZERO ) THEN
        THRESH = EPS
    ELSE
        THRESH = MAX( EPS*ANORM, SAFMIN )
    END IF
    DO 30 I = 1, K
        SEP( I ) = MAX( SEP( I ), THRESH )
30  CONTINUE

*
    RETURN
*
*   End of DDISNA
*

```

END

— LAPACK ddisna —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun ddisna (job m n d sep info)
    (declare (type (simple-array double-float (*)) sep d)
              (type fixnum info n m)
              (type character job))
    (f2cl-lib:with-multi-array-data
      ((job character job-%data% job-%offset%)
       (d double-float d-%data% d-%offset%)
       (sep double-float sep-%data% sep-%offset%))
      (prog ((anorm 0.0) (eps 0.0) (newgap 0.0) (oldgap 0.0) (safmin 0.0)
              (thresh 0.0) (i 0) (k 0) (decr nil) (eigen nil) (incr nil)
              (left nil) (right nil) (sing nil))
        (declare (type (double-float) anorm eps newgap oldgap safmin thresh)
                  (type fixnum i k)
                  (type (member t nil) decr eigen incr left right sing))
        (setf info 0)
        (setf eigen (char-equal job #\E))
        (setf left (char-equal job #\L))
        (setf right (char-equal job #\R))
        (setf sing (or left right))
        (cond
          (eigen
            (setf k m))
          (sing
            (setf k (min (the fixnum m) (the fixnum n)))))
        (cond
          ((and (not eigen) (not sing))
            (setf info -1))
          (< m 0)
            (setf info -2))
          (< k 0)
            (setf info -3))
          (t
            (setf incr t)
            (setf decr t)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
              (tagbody
                (if incr
                  (setf incr
                     (and incr

```

```

(=<
  (f2c1-lib:fref d-%data%
    (i)
    ((1 *))
    d-%offset%)
  (f2c1-lib:fref d-%data%
    ((f2c1-lib:int-add i 1))
    ((1 *))
    d-%offset%))))))

(if decr
  (setf decr
    (and decr
      (>=
        (f2c1-lib:fref d-%data%
          (i)
          ((1 *))
          d-%offset%)
        (f2c1-lib:fref d-%data%
          ((f2c1-lib:int-add i 1))
          ((1 *))
          d-%offset%)))))))

(cond
  ((and sing (> k 0))
    (if incr
      (setf incr
        (and incr
          (<= zero
            (f2c1-lib:fref d-%data%
              (1)
              ((1 *))
              d-%offset%))))))

    (if decr
      (setf decr
        (and decr
          (>=
            (f2c1-lib:fref d-%data% (k) ((1 *)) d-%offset%)
            zero))))))

    (if (not (or incr decr)) (setf info -4))))

(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DDISNA" (f2c1-lib:int-sub info))
    (go end_label)))
(if (= k 0) (go end_label))
(cond
  ((= k 1)
    (setf (f2c1-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
      (dlamch "0")))
  (t

```

```

(setf oldgap
  (abs
    (- (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%)
      (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
(setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%) oldgap)
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf newgap
      (abs
        (-
          (f2cl-lib:fref d-%data%
            ((f2cl-lib:int-add i 1))
            ((1 *))
            d-%offset%)
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
      (setf (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
        (min oldgap newgap))
      (setf oldgap newgap)))
    (setf (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%) oldgap)))
(cond
  (sing
    (cond
      ((or (and left (> m n)) (and right (< m n)))
        (if incr
          (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
            (min
              (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
              (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
          (if decr
            (setf (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%)
              (min
                (f2cl-lib:fref sep-%data% (k) ((1 *)) sep-%offset%)
                (f2cl-lib:fref d-%data%
                  (k)
                  ((1 *))
                  d-%offset%))))))
        (setf eps (dlamch "E"))
        (setf safmin (dlamch "S"))
        (setf anorm
          (max (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
              (abs (f2cl-lib:fref d-%data% (k) ((1 *)) d-%offset%))))
        (cond
          ((= anorm zero)
            (setf thresh eps))
          (t
            (setf thresh (max (* eps anorm) safmin))))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i k) nil)
          (tagbody

```

```

      (setf (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
            (max (f2cl-lib:fref sep-%data% (i) ((1 *)) sep-%offset%)
                  thresh))))
end_label
  (return (values nil nil nil nil nil info))))))

```

dgebak LAPACK

— dgebak.input —

```

)set break resume
)sys rm -f dgebak.output
)spool dgebak.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgebak.help —

```
=====
dgebak examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGEBAK - the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by DGEBA

SYNOPSIS

SUBROUTINE DGEBAK(JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV, INFO)

CHARACTER JOB, SIDE

INTEGER IHI, ILO, INFO, LDV, M, N

DOUBLE PRECISION SCALE(*), V(LDV, *)

PURPOSE

DGEBAK forms the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by DGEBAL.

ARGUMENTS

JOB (input) CHARACTER*1
Specifies the type of backward transformation required: = 'N', do nothing, return immediately; = 'P', do backward transformation for permutation only; = 'S', do backward transformation for scaling only; = 'B', do backward transformations for both permutation and scaling. JOB must be the same as the argument JOB supplied to DGEBAL.

SIDE (input) CHARACTER*1
= 'R': V contains right eigenvectors;
= 'L': V contains left eigenvectors.

N (input) INTEGER
The number of rows of the matrix V. N >= 0.

ILO (input) INTEGER
IHI (input) INTEGER The integers ILO and IHI determined by DGEBAL. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

SCALE (input) DOUBLE PRECISION array, dimension (N)
Details of the permutation and scaling factors, as returned by DGEBAL.

M (input) INTEGER
The number of columns of the matrix V. M >= 0.

V (input/output) DOUBLE PRECISION array, dimension (LDV,M)
On entry, the matrix of right or left eigenvectors to be transformed, as returned by DHSEIN or DTREVC. On exit, V is overwritten by the transformed eigenvectors.

LDV (input) INTEGER
The leading dimension of the array V. LDV >= max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value.

— dgebak.f —

```

      SUBROUTINE DGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV,
$                      INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  September 30, 1994
*
*  .. Scalar Arguments ..
*  CHARACTER          JOB, SIDE
*  INTEGER            IHI, ILO, INFO, LDV, M, N
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION   SCALE( * ), V( LDV, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
*  DOUBLE PRECISION   ONE
*  PARAMETER          ( ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL            LEFTV, RIGHTV
*  INTEGER            I, II, K
*  DOUBLE PRECISION   S
*
*  ..
*  .. External Functions ..
*  LOGICAL            LSAME
*  EXTERNAL           LSAME
*
*  ..
*  .. External Subroutines ..
*  EXTERNAL           DSCAL, DSWAP, XERBLA
*
*  ..
*  .. Intrinsic Functions ..
*  INTRINSIC          MAX, MIN
*
*  ..
*  .. Executable Statements ..
*
*  Decode and Test the input parameters
*
*  RIGHTV = LSAME( SIDE, 'R' )
*  LEFTV = LSAME( SIDE, 'L' )
*
*
*  INFO = 0

```

```

      IF( .NOT.LSAME( JOB, 'N' ) .AND. .NOT.LSAME( JOB, 'P' ) .AND.
$      .NOT.LSAME( JOB, 'S' ) .AND. .NOT.LSAME( JOB, 'B' ) ) THEN
          INFO = -1
      ELSE IF( .NOT.RIGHTV .AND. .NOT.LEFTV ) THEN
          INFO = -2
      ELSE IF( N.LT.0 ) THEN
          INFO = -3
      ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
          INFO = -4
      ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
          INFO = -5
      ELSE IF( M.LT.0 ) THEN
          INFO = -7
      ELSE IF( LDV.LT.MAX( 1, N ) ) THEN
          INFO = -9
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DGEBAK', -INFO )
          RETURN
      END IF
*
*      Quick return if possible
*
      IF( N.EQ.0 )
$      RETURN
      IF( M.EQ.0 )
$      RETURN
      IF( LSAME( JOB, 'N' ) )
$      RETURN
*
      IF( ILO.EQ.IHI )
$      GO TO 30
*
*      Backward balance
*
      IF( LSAME( JOB, 'S' ) .OR. LSAME( JOB, 'B' ) ) THEN
*
          IF( RIGHTV ) THEN
              DO 10 I = ILO, IHI
                  S = SCALE( I )
                  CALL DSCAL( M, S, V( I, 1 ), LDV )
10          CONTINUE
          END IF
*
          IF( LEFTV ) THEN
              DO 20 I = ILO, IHI
                  S = ONE / SCALE( I )
                  CALL DSCAL( M, S, V( I, 1 ), LDV )
20          CONTINUE
          END IF

```

```

*
*      END IF
*
*      Backward permutation
*
*      For I = ILO-1 step -1 until 1,
*          IHI+1 step 1 until N do --
*
30      CONTINUE
      IF( LSAME( JOB, 'P' ) .OR. LSAME( JOB, 'B' ) ) THEN
          IF( RIGHTV ) THEN
              DO 40 II = 1, N
                  I = II
                  IF( I.GE.ILO .AND. I.LE.IHI )
$                      GO TO 40
                  IF( I.LT.ILO )
$                      I = ILO - II
                  K = SCALE( I )
                  IF( K.EQ.I )
$                      GO TO 40
                  CALL DSWAP( M, V( I, 1 ), LDV, V( K, 1 ), LDV )
40          CONTINUE
          END IF
*
          IF( LEFTV ) THEN
              DO 50 II = 1, N
                  I = II
                  IF( I.GE.ILO .AND. I.LE.IHI )
$                      GO TO 50
                  IF( I.LT.ILO )
$                      I = ILO - II
                  K = SCALE( I )
                  IF( K.EQ.I )
$                      GO TO 50
                  CALL DSWAP( M, V( I, 1 ), LDV, V( K, 1 ), LDV )
50          CONTINUE
          END IF
      END IF
*
      RETURN
*
*      End of DGEBAK
*
      END

```

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgebak (job side n ilo ihi scale m v ldv info)
    (declare (type (simple-array double-float (*)) v scale)
              (type fixnum info ldv m ihi ilo n)
              (type character side job))
    (f2cl-lib:with-multi-array-data
      ((job character job-%data% job-%offset%)
       (side character side-%data% side-%offset%)
       (scale double-float scale-%data% scale-%offset%)
       (v double-float v-%data% v-%offset%))
      (prog ((s 0.0) (i 0) (ii 0) (k 0) (leftv nil) (rightv nil))
        (declare (type (double-float) s)
                  (type fixnum i ii k)
                  (type (member t nil) leftv rightv))
        (setf rightv (char-equal side #\R))
        (setf leftv (char-equal side #\L))
        (setf info 0)
        (cond
          ((and (not (char-equal job #\N))
                 (not (char-equal job #\P))
                 (not (char-equal job #\S))
                 (not (char-equal job #\B)))
           (setf info -1))
          ((and (not rightv) (not leftv))
           (setf info -2))
          ((< n 0)
           (setf info -3))
          ((or (< ilo 1)
                (> ilo
                  (max (the fixnum 1) (the fixnum n))))
           (setf info -4))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
           (setf info -5))
          ((< m 0)
           (setf info -7))
          ((< ldv (max (the fixnum 1) (the fixnum n)))
           (setf info -9)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGEBAK" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (= n 0) (go end_label))
        (if (= m 0) (go end_label))
        (if (char-equal job #\N) (go end_label))
        (if (= ilo ihi) (go label30))

```

```

(cond
  ((or (char-equal job #\S) (char-equal job #\B))
    (cond
      (rightv
        (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
          ((> i ihi) nil)
          (tagbody
            (setf s
              (f2cl-lib:fref scale-%data%
                (i)
                ((1 *))
                scale-%offset%)))
            (dscal m s
              (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
              ldv))))))
    (cond
      (leftv
        (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
          ((> i ihi) nil)
          (tagbody
            (setf s
              (/ one
                (f2cl-lib:fref scale-%data%
                  (i)
                  ((1 *))
                  scale-%offset%)))
            (dscal m s
              (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
              ldv)))))))))
label30
(cond
  ((or (char-equal job #\P) (char-equal job #\B))
    (cond
      (rightv
        (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
          ((> ii n) nil)
          (tagbody
            (setf i ii)
            (if (and (>= i ilo) (<= i ihi)) (go label40))
            (if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
            (setf k
              (f2cl-lib:int
                (f2cl-lib:fref scale-%data%
                  (i)
                  ((1 *))
                  scale-%offset%)))
            (if (= k i) (go label40))
            (dswap m
              (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
              ldv

```

```

                                (f2cl-lib:array-slice v double-float (k 1) ((1 ldv) (1 *)))
                                ldv)
label40))))
  (cond
    (leftv
      (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
                    ((> ii n) nil)
      (tagbody
        (setf i ii)
        (if (and (>= i ilo) (<= i ihi)) (go label50))
        (if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
        (setf k
              (f2cl-lib:int
                (f2cl-lib:fref scale-%data%
                              (i)
                              ((1 *))
                              scale-%offset%)))
        (if (= k i) (go label50))
        (dswap m
              (f2cl-lib:array-slice v double-float (i 1) ((1 ldv) (1 *)))
              ldv
              (f2cl-lib:array-slice v double-float (k 1) ((1 ldv) (1 *)))
              ldv)
        label50))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil info))))))

```

dgebal LAPACK

— dgebal.input —

```

)set break resume
)sys rm -f dgebal.output
)spool dgebal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgebal.help —

```
=====
dgebal examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGEBAL - a general real matrix A

SYNOPSIS

```
SUBROUTINE DGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
```

```
      CHARACTER      JOB
```

```
      INTEGER        IHI, ILO, INFO, LDA, N
```

```
      DOUBLE         PRECISION A( LDA, * ), SCALE( * )
```

PURPOSE

DGEBAL balances a general real matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

ARGUMENTS

JOB (input) CHARACTER*1
Specifies the operations to be performed on A:
= 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for
i = 1,...,N; = 'P': permute only;
= 'S': scale only;
= 'B': both permute and scale.

N (input) INTEGER
The order of the matrix A. N >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the input matrix A. On exit, A is overwritten by the balanced matrix. If JOB = 'N', A is not referenced. See Further Details. LDA (input) INTEGER The leading dimension of the array A. LDA >= max(1,N).

ILO (output) INTEGER

IHI (output) INTEGER ILO and IHI are set to integers such that on exit $A(i,j) = 0$ if $i > j$ and $j = 1, \dots, ILO-1$ or $i = IHI+1, \dots, N$. If $JOB = 'N'$ or $'S'$, $ILO = 1$ and $IHI = N$.

SCALE (output) DOUBLE PRECISION array, dimension (N)
 Details of the permutations and scaling factors applied to A. If $P(j)$ is the index of the row and column interchanged with row and column j and $D(j)$ is the scaling factor applied to row and column j , then $SCALE(j) = P(j)$ for $j = 1, \dots, ILO-1$ and $SCALE(j) = D(j)$ for $j = ILO, \dots, IHI$ and $SCALE(j) = P(j)$ for $j = IHI+1, \dots, N$. The order in which the interchanges are made is N to $IHI+1$, then 1 to $ILO-1$.

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if $INFO = -i$, the i -th argument had an illegal value.

FURTHER DETAILS

The permutations consist of row and column interchanges which put the matrix in the form

$$P A P = \begin{pmatrix} T1 & X & Y \\ 0 & B & Z \\ 0 & 0 & T2 \end{pmatrix}$$

where $T1$ and $T2$ are upper triangular matrices whose eigenvalues lie along the diagonal. The column indices ILO and IHI mark the starting and ending columns of the submatrix B . Balancing consists of applying a diagonal similarity transformation $\text{inv}(D) * B * D$ to make the 1-norms of each row of B and its corresponding column nearly equal. The output matrix is

$$\begin{pmatrix} T1 & X*D & Y \\ 0 & \text{inv}(D)*B*D & \text{inv}(D)*Z \\ 0 & 0 & T2 \end{pmatrix}.$$

Information about the permutations P and the diagonal matrix D is returned in the vector $SCALE$.

This subroutine is based on the EISPACK routine `BALANC`.

Modified by Tzu-Yi Chen, Computer Science Division, University of California at Berkeley, USA


```

      SUBROUTINE DGEBA( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      CHARACTER          JOB
      INTEGER            IHI, ILO, INFO, LDA, N
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), SCALE( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION   ZERO, ONE
      PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0 )
      DOUBLE PRECISION   SCLFAC
      PARAMETER          ( SCLFAC = 0.8D+1 )
      DOUBLE PRECISION   FACTOR
      PARAMETER          ( FACTOR = 0.95D+0 )
*
*  ..
*
*  .. Local Scalars ..
      LOGICAL            NOCONV
      INTEGER            I, ICA, IEXC, IRA, J, K, L, M
      DOUBLE PRECISION   C, CA, F, G, R, RA, S, SFMAX1, SFMAX2, SFMIN1,
$                        SFMIN2
*
*  ..
*
*  .. External Functions ..
      LOGICAL            LSAME
      INTEGER            IDAMAX
      DOUBLE PRECISION   DLAMCH
      EXTERNAL           LSAME, IDAMAX, DLAMCH
*
*  ..
*
*  .. External Subroutines ..
      EXTERNAL           DSCAL, DSWAP, XERBLA
*
*  ..
*
*  .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, MIN
*
*  ..
*
*  .. Executable Statements ..
*
*  Test the input parameters
*
      INFO = 0
      IF( .NOT.LSAME( JOB, 'N' ) .AND. .NOT.LSAME( JOB, 'P' ) .AND.

```

```

$      .NOT.LSAME( JOB, 'S' ) .AND. .NOT.LSAME( JOB, 'B' ) ) THEN
      INFO = -1
      ELSE IF( N.LT.0 ) THEN
        INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DGEBA', -INFO )
        RETURN
      END IF
*
      K = 1
      L = N
*
      IF( N.EQ.0 )
$      GO TO 210
*
      IF( LSAME( JOB, 'N' ) ) THEN
        DO 10 I = 1, N
          SCALE( I ) = ONE
10      CONTINUE
        GO TO 210
      END IF
*
      IF( LSAME( JOB, 'S' ) )
$      GO TO 120
*
*      Permutation to isolate eigenvalues if possible
*
      GO TO 50
*
*      Row and column exchange.
*
20 CONTINUE
      SCALE( M ) = J
      IF( J.EQ.M )
$      GO TO 30
*
      CALL DSWAP( L, A( 1, J ), 1, A( 1, M ), 1 )
      CALL DSWAP( N-K+1, A( J, K ), LDA, A( M, K ), LDA )
*
30 CONTINUE
      GO TO ( 40, 80 )IEXC
*
*      Search for rows isolating an eigenvalue and push them down.
*
40 CONTINUE
      IF( L.EQ.1 )
$      GO TO 210

```

```

      L = L - 1
*
50  CONTINUE
      DO 70 J = L, 1, -1
*
          DO 60 I = 1, L
              IF( I.EQ.J )
$                  GO TO 60
              IF( A( J, I ).NE.ZERO )
$                  GO TO 70
60      CONTINUE
*
          M = L
          IEXC = 1
          GO TO 20
70  CONTINUE
*
      GO TO 90
*
*      Search for columns isolating an eigenvalue and push them left.
*
80  CONTINUE
      K = K + 1
*
90  CONTINUE
      DO 110 J = K, L
*
          DO 100 I = K, L
              IF( I.EQ.J )
$                  GO TO 100
              IF( A( I, J ).NE.ZERO )
$                  GO TO 110
100     CONTINUE
*
          M = K
          IEXC = 2
          GO TO 20
110  CONTINUE
*
120  CONTINUE
      DO 130 I = K, L
          SCALE( I ) = ONE
130  CONTINUE
*
      IF( LSAME( JOB, 'P' ) )
$          GO TO 210
*
*      Balance the submatrix in rows K to L.
*
*      Iterative loop for norm reduction

```

```

*
      SFMIN1 = DLAMCH( 'S' ) / DLAMCH( 'P' )
      SFMAX1 = ONE / SFMIN1
      SFMIN2 = SFMIN1*SCLFAC
      SFMAX2 = ONE / SFMIN2
140  CONTINUE
      NOCONV = .FALSE.
*
      DO 200 I = K, L
        C = ZERO
        R = ZERO
*
        DO 150 J = K, L
          IF( J.EQ.I )
            $      GO TO 150
          C = C + ABS( A( J, I ) )
          R = R + ABS( A( I, J ) )
150  CONTINUE
          ICA = IDAMAX( L, A( 1, I ), 1 )
          CA = ABS( A( ICA, I ) )
          IRA = IDAMAX( N-K+1, A( I, K ), LDA )
          RA = ABS( A( I, IRA+K-1 ) )
*
*      Guard against zero C or R due to underflow.
*
          IF( C.EQ.ZERO .OR. R.EQ.ZERO )
            $      GO TO 200
          G = R / SCLFAC
          F = ONE
          S = C + R
160  CONTINUE
          IF( C.GE.G .OR. MAX( F, C, CA ).GE.SFMAX2 .OR.
            $      MIN( R, G, RA ).LE.SFMIN2 )GO TO 170
          F = F*SCLFAC
          C = C*SCLFAC
          CA = CA*SCLFAC
          R = R / SCLFAC
          G = G / SCLFAC
          RA = RA / SCLFAC
          GO TO 160
*
170  CONTINUE
          G = C / SCLFAC
180  CONTINUE
          IF( G.LT.R .OR. MAX( R, RA ).GE.SFMAX2 .OR.
            $      MIN( F, C, G, CA ).LE.SFMIN2 )GO TO 190
          F = F / SCLFAC
          C = C / SCLFAC
          G = G / SCLFAC
          CA = CA / SCLFAC

```

```

        R = R*SCLFAC
        RA = RA*SCLFAC
        GO TO 180
*
*       Now balance.
*
190    CONTINUE
        IF( ( C+R ).GE.FACTOR*S )
$      GO TO 200
        IF( F.LT.ONE .AND. SCALE( I ).LT.ONE ) THEN
            IF( F*SCALE( I ).LE.SFMIN1 )
$          GO TO 200
        END IF
        IF( F.GT.ONE .AND. SCALE( I ).GT.ONE ) THEN
            IF( SCALE( I ).GE.SFMAX1 / F )
$          GO TO 200
        END IF
        G = ONE / F
        SCALE( I ) = SCALE( I )*F
        NOCONV = .TRUE.
*
        CALL DSCAL( N-K+1, G, A( I, K ), LDA )
        CALL DSCAL( L, F, A( 1, I ), 1 )
*
200    CONTINUE
*
        IF( NOCONV )
$      GO TO 140
*
210    CONTINUE
        ILO = K
        IHI = L
*
        RETURN
*
*       End of DGEBAL
*
END

```

— LAPACK dgebal —

```

(let* ((zero 0.0) (one 1.0) (sclfac 8.0) (factor 0.95))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 8.0 8.0) sclfac)
            (type (double-float 0.95 0.95) factor))

```

```

(defun dgebal (job n a lda ilo ihi scale info)
  (declare (type (simple-array double-float (*)) scale a)
    (type fixnum info ihi ilo lda n)
    (type character job))
  (f2cl-lib:with-multi-array-data
    ((job character job-%data% job-%offset%)
     (a double-float a-%data% a-%offset%)
     (scale double-float scale-%data% scale-%offset%))
    (prog ((c 0.0) (ca 0.0) (f 0.0) (g 0.0) (r 0.0) (ra 0.0) (s 0.0)
      (sfmax1 0.0) (sfmax2 0.0) (sfmin1 0.0) (sfmin2 0.0) (i 0) (ica 0)
      (iexc 0) (ira 0) (j 0) (k 0) (l 0) (m 0) (noconv nil))
      (declare (type (double-float) c ca f g r ra s sfmax1 sfmax2 sfmin1
        sfmin2)
        (type fixnum i ica iexc ira j k l m)
        (type (member t nil) noconv))
        (setf info 0)
        (cond
          ((and (not (char-equal job #\N))
            (not (char-equal job #\P))
            (not (char-equal job #\S))
            (not (char-equal job #\B))))
            (setf info -1))
          ((< n 0)
            (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum n)))
            (setf info -4)))
          (cond
            ((/= info 0)
              (error
                " ** On entry to ~a parameter number ~a had an illegal value~%"
                "DGEBAL" (f2cl-lib:int-sub info))
              (go end_label)))
            (setf k 1)
            (setf l n)
            (if (= n 0) (go label210))
            (cond
              ((char-equal job #\N)
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  (> i n) nil)
                (tagbody
                  (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
                    one)))
                (go label210)))
              (if (char-equal job #\S) (go label120))
              (go label50)
            )
          )
        label20
        (setf (f2cl-lib:fref scale-%data% (m) ((1 *)) scale-%offset%)
          (coerce (the fixnum j) 'double-float))
        (if (= j m) (go label30))
        (dswap 1 (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *))) 1

```

```

        (f2cl-lib:array-slice a double-float (1 m) ((1 lda) (1 *))) 1)
      (dswap (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
        (f2cl-lib:array-slice a double-float (j k) ((1 lda) (1 *))) lda
        (f2cl-lib:array-slice a double-float (m k) ((1 lda) (1 *))) lda)
label30
      (f2cl-lib:computed-goto (label40 label80) iexc)
label40
      (if (= 1 1) (go label210))
      (setf 1 (f2cl-lib:int-sub 1 1))
label50
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j 1) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i 1) nil)
            (tagbody
              (if (= i j) (go label60))
              (if
                (/= (f2cl-lib:fref a-%data% (j i) ((1 lda) (1 *)) a-%offset%)
                  zero)
                (go label70))
              label60))
          (setf m 1)
          (setf iexc 1)
          (go label20)
          label70))
        (go label90)
label80
      (setf k (f2cl-lib:int-add k 1))
label90
      (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
        (> j 1) nil)
        (tagbody
          (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
            (> i 1) nil)
            (tagbody
              (if (= i j) (go label100))
              (if
                (/= (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
                  zero)
                (go label110))
              label100))
          (setf m k)
          (setf iexc 2)
          (go label20)
          label110))
label110
label120
      (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
        (> i 1) nil)
        (tagbody

```

```

        (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset% one)))
      (if (char-equal job #\P) (go label210))
      (setf sfmin1 (/ (dlamch "S") (dlamch "P")))
      (setf sfmax1 (/ one sfmin1))
      (setf sfmin2 (* sfmin1 sclfac))
      (setf sfmax2 (/ one sfmin2))
label1140
      (setf noconv nil)
      (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
        ((> i 1) nil)
        (tagbody
          (setf c zero)
          (setf r zero)
          (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
            ((> j 1) nil)
            (tagbody
              (if (= j i) (go label1150))
              (setf c
                (+ c
                  (abs
                    (f2cl-lib:fref a-%data%
                      (j i)
                      ((1 lda) (1 *))
                      a-%offset%))))))
              (setf r
                (+ r
                  (abs
                    (f2cl-lib:fref a-%data%
                      (i j)
                      ((1 lda) (1 *))
                      a-%offset%))))))
            label1150))
          (setf ica
            (idamax 1
              (f2cl-lib:array-slice a
                double-float
                (1 i)
                ((1 lda) (1 *)))
              1))
          (setf ca
            (abs
              (f2cl-lib:fref a-%data%
                (ica i)
                ((1 lda) (1 *))
                a-%offset%)))
          (setf ira
            (idamax (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
              (f2cl-lib:array-slice a
                double-float
                (i k)

```



```

                                ((1 lda) (1 *)))
                                lda))
(setf ra
  (abs
    (f2cl-lib:fref a-%data%
      (i
        (f2cl-lib:int-sub (f2cl-lib:int-add ira k)
          1))
        ((1 lda) (1 *))
        a-%offset%)))
(if (or (= c zero) (= r zero)) (go label200))
(setf g (/ r sclfac))
(setf f one)
(setf s (+ c r))
label160
  (if (or (>= c g) (>= (max f c ca) sfmax2) (<= (min r g ra) sfmin2))
    (go label170))
  (setf f (* f sclfac))
  (setf c (* c sclfac))
  (setf ca (* ca sclfac))
  (setf r (/ r sclfac))
  (setf g (/ g sclfac))
  (setf ra (/ ra sclfac))
  (go label160)
label170
  (setf g (/ c sclfac))
label180
  (if (or (< g r) (>= (max r ra) sfmax2) (<= (min f c g ca) sfmin2))
    (go label190))
  (setf f (/ f sclfac))
  (setf c (/ c sclfac))
  (setf g (/ g sclfac))
  (setf ca (/ ca sclfac))
  (setf r (* r sclfac))
  (setf ra (* ra sclfac))
  (go label180)
label190
  (if (>= (+ c r) (* factor s)) (go label200))
  (cond
    ((and (< f one) (< (f2cl-lib:fref scale (i) ((1 *))) one))
      (if
        (<=
          (* f (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%))
          sfmin1)
        (go label200))))
    (cond
      ((and (> f one) (> (f2cl-lib:fref scale (i) ((1 *))) one))
        (if
          (>= (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
            (/ sfmax1 f))

```

```

      (go label200)))
    (setf g (/ one f))
    (setf (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
          (* (f2cl-lib:fref scale-%data% (i) ((1 *)) scale-%offset%)
             f))
    (setf noconv t)
    (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1) g
          (f2cl-lib:array-slice a double-float (i k) ((1 lda) (1 *))) lda)
    (dscal 1 f
          (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1)
  label200))
  (if noconv (go label140))
  label210
    (setf ilo k)
    (setf ihi l)
  end_label
  (return (values nil nil nil nil ilo ihi nil info))))))

```

dgebd2 LAPACK

— dgebd2.input —

```

)set break resume
)sys rm -f dgebd2.output
)spool dgebd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgebd2.help —

```

=====
dgebd2 examples
=====

=====
Man Page Details
=====

```

NAME

DGEBD2 - a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation

SYNOPSIS

SUBROUTINE DGEBD2(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO)

INTEGER INFO, LDA, M, N

DOUBLE PRECISION A(LDA, *), D(*), E(*), TAUP(*),
TAUQ(*), WORK(*)

PURPOSE

DGEBD2 reduces a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: $Q' * A * P = B$.

If $m \geq n$, B is upper bidiagonal; if $m < n$, B is lower bidiagonal.

ARGUMENTS

- M (input) INTEGER
The number of rows in the matrix A. $M \geq 0$.
- N (input) INTEGER
The number of columns in the matrix A. $N \geq 0$.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the m by n general matrix to be reduced. On exit, if $m \geq n$, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if $m < n$, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,M)$.
- D (output) DOUBLE PRECISION array, dimension (min(M,N))
The diagonal elements of the bidiagonal matrix B: $D(i) = A(i,i)$.
- E (output) DOUBLE PRECISION array, dimension (min(M,N)-1)
The off-diagonal elements of the bidiagonal matrix B: if $m \geq$

n , $E(i) = A(i, i+1)$ for $i = 1, 2, \dots, n-1$; if $m < n$, $E(i) = A(i+1, i)$ for $i = 1, 2, \dots, m-1$.

TAUQ (output) DOUBLE PRECISION array dimension $(\min(M, N))$
 The scalar factors of the elementary reflectors which represent the orthogonal matrix Q . See Further Details. **TAUP** (output) DOUBLE PRECISION array, dimension $(\min(M, N))$ The scalar factors of the elementary reflectors which represent the orthogonal matrix P . See Further Details. **WORK** (workspace) DOUBLE PRECISION array, dimension $(\max(M, N))$

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if $\text{INFO} = -i$, the i -th argument had an illegal value.

FURTHER DETAILS

The matrices Q and P are represented as products of elementary reflectors:

If $m \geq n$,

$$Q = H(1) H(2) \dots H(n) \quad \text{and} \quad P = G(1) G(2) \dots G(n-1)$$

Each $H(i)$ and $G(i)$ has the form:

$$H(i) = I - \tau_q * v * v' \quad \text{and} \quad G(i) = I - \tau_p * u * u'$$

where τ_q and τ_p are real scalars, and v and u are real vectors; $v(1:i-1) = 0$, $v(i) = 1$, and $v(i+1:m)$ is stored on exit in $A(i+1:m, i)$; $u(1:i) = 0$, $u(i+1) = 1$, and $u(i+2:n)$ is stored on exit in $A(i, i+2:n)$; τ_q is stored in $\text{TAUQ}(i)$ and τ_p in $\text{TAUP}(i)$.

If $m < n$,

$$Q = H(1) H(2) \dots H(m-1) \quad \text{and} \quad P = G(1) G(2) \dots G(m)$$

Each $H(i)$ and $G(i)$ has the form:

$$H(i) = I - \tau_q * v * v' \quad \text{and} \quad G(i) = I - \tau_p * u * u'$$

where τ_q and τ_p are real scalars, and v and u are real vectors; $v(1:i) = 0$, $v(i+1) = 1$, and $v(i+2:m)$ is stored on exit in $A(i+2:m, i)$; $u(1:i-1) = 0$, $u(i) = 1$, and $u(i+1:n)$ is stored on exit in $A(i, i+1:n)$; τ_q is stored in $\text{TAUQ}(i)$ and τ_p in $\text{TAUP}(i)$.

The contents of A on exit are illustrated by the following examples:

$m = 6$ and $n = 5$ ($m > n$):

(d e u1 u1 u1)

$m = 5$ and $n = 6$ ($m < n$):

(d u1 u1 u1 u1 u1)

| | |
|--------------------|---------------------|
| (v1 d e u2 u2) | (e d u2 u2 u2 u2) |
| (v1 v2 d e u3) | (v1 e d u3 u3 u3) |
| (v1 v2 v3 d e) | (v1 v2 e d u4 u4) |
| (v1 v2 v3 v4 d) | (v1 v2 v3 e d u5) |
| (v1 v2 v3 v4 v5) | |

where d and e denote diagonal and off-diagonal elements of B, vi denotes an element of the vector defining H(i), and ui an element of the vector defining G(i).

— dgebd2.f —

```

SUBROUTINE DGEBD2( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, INFO )
*
* -- LAPACK routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   February 29, 1992
*
* .. Scalar Arguments ..
*   INTEGER          INFO, LDA, M, N
* ..
* .. Array Arguments ..
*   DOUBLE PRECISION A( LDA, * ), D( * ), E( * ), TAUP( * ),
*   $                TAUQ( * ), WORK( * )
* ..
*
* =====
*
* .. Parameters ..
*   DOUBLE PRECISION ZERO, ONE
*   PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
* ..
* .. Local Scalars ..
*   INTEGER          I
* ..
* .. External Subroutines ..
*   EXTERNAL         DLARF, DLARFG, XERBLA
* ..
* .. Intrinsic Functions ..
*   INTRINSIC        MAX, MIN
* ..
* .. Executable Statements ..
*
*   Test the input parameters
*

```

```

INFO = 0
IF( M.LT.0 ) THEN
    INFO = -1
ELSE IF( N.LT.0 ) THEN
    INFO = -2
ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
    INFO = -4
END IF
IF( INFO.LT.0 ) THEN
    CALL XERBLA( 'DGEBD2', -INFO )
    RETURN
END IF

*
IF( M.GE.N ) THEN
*
*   Reduce to upper bidiagonal form
*
DO 10 I = 1, N
*
*   Generate elementary reflector H(i) to annihilate A(i+1:m,i)
*
CALL DLARFG( M-I+1, A( I, I ), A( MIN( I+1, M ), I ), 1,
$           TAUQ( I ) )
D( I ) = A( I, I )
A( I, I ) = ONE
*
*   Apply H(i) to A(i:m,i+1:n) from the left
*
CALL DLARF( 'Left', M-I+1, N-I, A( I, I ), 1, TAUQ( I ),
$           A( I, I+1 ), LDA, WORK )
A( I, I ) = D( I )
*
IF( I.LT.N ) THEN
*
*   Generate elementary reflector G(i) to annihilate
*   A(i,i+2:n)
*
CALL DLARFG( N-I, A( I, I+1 ), A( I, MIN( I+2, N ) ),
$           LDA, TAUP( I ) )
E( I ) = A( I, I+1 )
A( I, I+1 ) = ONE
*
*   Apply G(i) to A(i+1:m,i+1:n) from the right
*
CALL DLARF( 'Right', M-I, N-I, A( I, I+1 ), LDA,
$           TAUP( I ), A( I+1, I+1 ), LDA, WORK )
A( I, I+1 ) = E( I )
ELSE
    TAUP( I ) = ZERO
END IF

```

```

10    CONTINUE
      ELSE
*
*        Reduce to lower bidiagonal form
*
      DO 20 I = 1, M
*
*        Generate elementary reflector G(i) to annihilate A(i,i+1:n)
*
        CALL DLARFG( N-I+1, A( I, I ), A( I, MIN( I+1, N ) ), LDA,
$          TAUP( I ) )
        D( I ) = A( I, I )
        A( I, I ) = ONE
*
*        Apply G(i) to A(i+1:m,i:n) from the right
*
        CALL DLARF( 'Right', M-I, N-I+1, A( I, I ), LDA, TAUP( I ),
$          A( MIN( I+1, M ), I ), LDA, WORK )
        A( I, I ) = D( I )
*
        IF( I.LT.M ) THEN
*
*        Generate elementary reflector H(i) to annihilate
*        A(i+2:m,i)
*
        CALL DLARFG( M-I, A( I+1, I ), A( MIN( I+2, M ), I ), 1,
$          TAUQ( I ) )
        E( I ) = A( I+1, I )
        A( I+1, I ) = ONE
*
*        Apply H(i) to A(i+1:m,i+1:n) from the left
*
        CALL DLARF( 'Left', M-I, N-I, A( I+1, I ), 1, TAUQ( I ),
$          A( I+1, I+1 ), LDA, WORK )
        A( I+1, I ) = E( I )
        ELSE
          TAUQ( I ) = ZERO
        END IF
20    CONTINUE
      END IF
      RETURN
*
*    End of DGEBD2
*
      END

```

— LAPACK dgebd2 —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgebd2 (m n a lda d e tauq taup work info)
    (declare (type (simple-array double-float (*)) work taup tauq e d a)
              (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (tauq double-float tauq-%data% tauq-%offset%)
       (taup double-float taup-%data% taup-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0))
        (declare (type fixnum i))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info -1))
          ((< n 0)
            (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info -4)))
        (cond
          ((< info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DGEBD2" (f2cl-lib:int-sub info))
            (go end_label)))
        (cond
          ((>= m n)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          ((> i n) nil)
            (tagbody
              (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                      (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                      (f2cl-lib:array-slice a
                                            double-float
                                            ((min (f2cl-lib:int-add i 1) m) i)
                                            ((1 lda) (1 *)))
                      1 (f2cl-lib:fref tauq-%data% (i i) ((1 *) tauq-%offset%)))
                (declare (ignore var-0 var-2 var-3))
                (setf (f2cl-lib:fref a-%data%
                                     (i i)
                                     ((1 lda) (1 *))
                                     a-%offset%)
                      var-1)
              var-1)
            )
          )
        )
      )
    )
  )

```



```
(setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
      var-4))
(setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
      (f2cl-lib:fref a-%data%
                      (i i)
                      ((1 lda) (1 *)))
      a-%offset%))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *))) a-%offset%)
one)
(dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
       (f2cl-lib:int-sub n i)
       (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
       (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
       (f2cl-lib:array-slice a
                               double-float
                               (i (f2cl-lib:int-add i 1))
                               ((1 lda) (1 *))))
lda work)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *))) a-%offset%)
(f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
(cond
 (< i n)
 (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
   (dlarfg (f2cl-lib:int-sub n i)
            (f2cl-lib:fref a-%data%
                          (i (f2cl-lib:int-add i 1))
                          ((1 lda) (1 *)))
            a-%offset%)
   (f2cl-lib:array-slice a
                         double-float
                         (i
                          (min
                           (the fixnum
                            (f2cl-lib:int-add i 2))
                           (the fixnum n)))
                         ((1 lda) (1 *))))
   lda
   (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *)))
      a-%offset%)

      var-1)
(setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
      var-4))
(setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
      (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *)))
```

```

                                a-%offset%))
(setf (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *))
                    a-%offset%)
      one)
(dlarf "Right" (f2cl-lib:int-sub m i) (f2cl-lib:int-sub n i)
      (f2cl-lib:array-slice a
                            double-float
                            (i (f2cl-lib:int-add i 1))
                            ((1 lda) (1 *)))
      lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
      (f2cl-lib:array-slice a
                            double-float
                            ((+ i 1) (f2cl-lib:int-add i 1))
                            ((1 lda) (1 *)))

      lda work)
(setf (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *))
                    a-%offset%)
      (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)))
(t
  (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
        zero))))))
(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                (> i m) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
            (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
            (f2cl-lib:array-slice a
                                  double-float
                                  (i
                                    (min
                                      (the fixnum
                                        (f2cl-lib:int-add i 1))
                                      (the fixnum n)))
                                  ((1 lda) (1 *)))
            lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
      (declare (ignore var-0 var-2 var-3))
      (setf (f2cl-lib:fref a-%data%
                          (i i)
                          ((1 lda) (1 *))
                          a-%offset%)
            var-1)
      (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
            var-4))
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)

```

```

(f2cl-lib:fref a-%data%
  (i i)
  ((1 lda) (1 *))
  a-%offset%)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
      one)
(dlarf "Right" (f2cl-lib:int-sub m i)
(f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
(f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
(f2cl-lib:array-slice a
  double-float
  ((min (f2cl-lib:int-add i 1) m) i)
  ((1 lda) (1 *)))
lda work)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
      (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
(cond
  ((< i m)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlarf (f2cl-lib:int-sub m i)
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add i 1) i)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:array-slice a
          double-float
          ((min (f2cl-lib:int-add i 2) m) i)
          ((1 lda) (1 *)))
          1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
      (declare (ignore var-0 var-2 var-3))
      (setf (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add i 1) i)
        ((1 lda) (1 *))
        a-%offset%)
        var-1)
      (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
        var-4))
    (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
      (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add i 1) i)
        ((1 lda) (1 *))
        a-%offset%))
    (setf (f2cl-lib:fref a-%data%
      ((f2cl-lib:int-add i 1) i)
      ((1 lda) (1 *))
      a-%offset%)
      one)
    (dlarf "Left" (f2cl-lib:int-sub m i) (f2cl-lib:int-sub n i)
      (f2cl-lib:array-slice a

```

```

                                double-float
                                ((+ i 1) i)
                                ((1 lda) (1 *)))
1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
(f2cl-lib:array-slice a
  double-float
  ((+ i 1) (f2cl-lib:int-add i 1))
  ((1 lda) (1 *)))
lda work)
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-add i 1) i)
  ((1 lda) (1 *))
  a-%offset%)
  (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)))
(t
  (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
    zero))))))
end_label
(return (values nil nil nil nil nil nil nil nil info))))

```

dgebrd LAPACK

— dgebrd.input —

```

)set break resume
)sys rm -f dgebrd.output
)spool dgebrd.output
)set message test on
)set message auto off
)clear all

```

```

)spool
)lisp (bye)

```

— dgebrd.help —

```

=====
dgebrd examples
=====
=====

```

Man Page Details

=====

NAME

DGEBRD - a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation

SYNOPSIS

SUBROUTINE DGEBRD(M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK, INFO)

INTEGER INFO, LDA, LWORK, M, N

DOUBLE PRECISION A(LDA, *), D(*), E(*), TAUP(*),
TAUQ(*), WORK(*)

PURPOSE

DGEBRD reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation: $Q^*T * A * P = B$.

If $m \geq n$, B is upper bidiagonal; if $m < n$, B is lower bidiagonal.

ARGUMENTS

- M (input) INTEGER
The number of rows in the matrix A. $M \geq 0$.
- N (input) INTEGER
The number of columns in the matrix A. $N \geq 0$.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N general matrix to be reduced. On exit, if $m \geq n$, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors; if $m < n$, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,M)$.
- D (output) DOUBLE PRECISION array, dimension (min(M,N))
The diagonal elements of the bidiagonal matrix B: $D(i) = A(i,i)$.

E (output) DOUBLE PRECISION array, dimension $(\min(M,N)-1)$
 The off-diagonal elements of the bidiagonal matrix B: if $m \geq n$, $E(i) = A(i,i+1)$ for $i = 1, 2, \dots, n-1$; if $m < n$, $E(i) = A(i+1,i)$ for $i = 1, 2, \dots, m-1$.

TAUQ (output) DOUBLE PRECISION array dimension $(\min(M,N))$
 The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details. **TAUP** (output) DOUBLE PRECISION array, dimension $(\min(M,N))$ The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details. **WORK** (workspace/output) DOUBLE PRECISION array, dimension $(\max(1,LWORK))$ On exit, if **INFO** = 0, **WORK(1)** returns the optimal **LWORK**.

LWORK (input) INTEGER
 The length of the array **WORK**. $LWORK \geq \max(1,M,N)$. For optimum performance $LWORK \geq (M+N)*NB$, where **NB** is the optimal blocksize.

If **LWORK** = -1, then a workspace query is assumed; the routine only calculates the optimal size of the **WORK** array, returns this value as the first entry of the **WORK** array, and no error message related to **LWORK** is issued by XERBLA.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if **INFO** = -i, the i-th argument had an illegal value.

FURTHER DETAILS

The matrices Q and P are represented as products of elementary reflectors:

If $m \geq n$,

$$Q = H(1) H(2) \dots H(n) \quad \text{and} \quad P = G(1) G(2) \dots G(n-1)$$

Each $H(i)$ and $G(i)$ has the form:

$$H(i) = I - \tau_q * v * v' \quad \text{and} \quad G(i) = I - \tau_p * u * u'$$

where τ_q and τ_p are real scalars, and v and u are real vectors; $v(1:i-1) = 0$, $v(i) = 1$, and $v(i+1:m)$ is stored on exit in $A(i+1:m,i)$; $u(1:i) = 0$, $u(i+1) = 1$, and $u(i+2:n)$ is stored on exit in $A(i,i+2:n)$; τ_q is stored in **TAUQ(i)** and τ_p in **TAUP(i)**.

If $m < n$,

$$Q = H(1) H(2) \dots H(m-1) \quad \text{and} \quad P = G(1) G(2) \dots G(m)$$

Each $H(i)$ and $G(i)$ has the form:

$$H(i) = I - \text{tauq} * v * v' \quad \text{and} \quad G(i) = I - \text{taup} * u * u'$$

where tauq and taup are real scalars, and v and u are real vectors;
 $v(1:i) = 0$, $v(i+1) = 1$, and $v(i+2:m)$ is stored on exit in $A(i+2:m,i)$;
 $u(1:i-1) = 0$, $u(i) = 1$, and $u(i+1:n)$ is stored on exit in $A(i,i+1:n)$;
tauq is stored in $\text{TAUQ}(i)$ and taup in $\text{TAUP}(i)$.

The contents of A on exit are illustrated by the following examples:

m = 6 and n = 5 (m > n):

```
( d  e  u1 u1 u1 )
( v1 d  e  u2 u2 )
( v1 v2 d  e  u3 )
( v1 v2 v3 d  e  )
( v1 v2 v3 v4 d  )
( v1 v2 v3 v4 v5 )
```

m = 5 and n = 6 (m < n):

```
( d  u1 u1 u1 u1 )
( e  d  u2 u2 u2 )
( v1 e  d  u3 u3 )
( v1 v2 e  d  u4 )
( v1 v2 v3 e  d  u5 )
```

where d and e denote diagonal and off-diagonal elements of B, vi denotes an element of the vector defining $H(i)$, and ui an element of the vector defining $G(i)$.

— dgebrd.f —

```
SUBROUTINE DGEBRD( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK,
$                INFO )
*
* -- LAPACK routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University
* June 30, 1999
*
* .. Scalar Arguments ..
INTEGER          INFO, LDA, LWORK, M, N
*
* ..
* .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), D( * ), E( * ), TAUP( * ),
$                TAUQ( * ), WORK( * )
*
* ..
*
* =====
*
* .. Parameters ..
DOUBLE PRECISION ONE
PARAMETER        ( ONE = 1.0D+0 )
```

```

*      ..
*      .. Local Scalars ..
      LOGICAL          LQUERY
      INTEGER          I, IINFO, J, LDWRKX, LDWRKY, LWKOPT, MINMN, NB,
$      NBMIN, NX
      DOUBLE PRECISION WS
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DGEBD2, DGEMM, DLABRD, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          DBLE, MAX, MIN
*
*      ..
*      .. External Functions ..
      INTEGER          ILAENV
      EXTERNAL          ILAENV
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters
*
      INFO = 0
      NB = MAX( 1, ILAENV( 1, 'DGEBRD', ' ', M, N, -1, -1 ) )
      LWKOPT = ( M+N )*NB
      WORK( 1 ) = DBLE( LWKOPT )
      LQUERY = ( LWORK.EQ.-1 )
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( LWORK.LT.MAX( 1, M, N ) .AND. .NOT.LQUERY ) THEN
         INFO = -10
      END IF
      IF( INFO.LT.0 ) THEN
         CALL XERBLA( 'DGEBRD', -INFO )
         RETURN
      ELSE IF( LQUERY ) THEN
         RETURN
      END IF
*
*      Quick return if possible
*
*
      MINMN = MIN( M, N )
      IF( MINMN.EQ.0 ) THEN
         WORK( 1 ) = 1
         RETURN
      END IF
*

```



```

      WS = MAX( M, N )
      LDWRKX = M
      LDWRKY = N
*
      IF( NB.GT.1 .AND. NB.LT.MINMN ) THEN
*
*       Set the crossover point NX.
*
      NX = MAX( NB, ILAENV( 3, 'DGEBRD', ' ', M, N, -1, -1 ) )
*
*       Determine when to switch from blocked to unblocked code.
*
      IF( NX.LT.MINMN ) THEN
        WS = ( M+N )*NB
        IF( LWORK.LT.WS ) THEN
*
*          Not enough work space for the optimal NB, consider using
*          a smaller block size.
*
          NBMIN = ILAENV( 2, 'DGEBRD', ' ', M, N, -1, -1 )
          IF( LWORK.GE.( M+N )*NBMIN ) THEN
            NB = LWORK / ( M+N )
          ELSE
            NB = 1
            NX = MINMN
          END IF
        END IF
      END IF
      ELSE
        NX = MINMN
      END IF
*
      DO 30 I = 1, MINMN - NX, NB
*
*       Reduce rows and columns i:i+nb-1 to bidiagonal form and return
*       the matrices X and Y which are needed to update the unreduced
*       part of the matrix
*
      CALL DLABRD( M-I+1, N-I+1, NB, A( I, I ), LDA, D( I ), E( I ),
$           TAUQ( I ), TAUP( I ), WORK, LDWRKX,
$           WORK( LDWRKX*NB+1 ), LDWRKY )
*
*       Update the trailing submatrix A(i+nb:m,i+nb:n), using an update
*       of the form  A := A - V*Y' - X*U'
*
      CALL DGEMM( 'No transpose', 'Transpose', M-I-NB+1, N-I-NB+1,
$           NB, -ONE, A( I+NB, I ), LDA,
$           WORK( LDWRKX*NB+NB+1 ), LDWRKY, ONE,
$           A( I+NB, I+NB ), LDA )
      CALL DGEMM( 'No transpose', 'No transpose', M-I-NB+1, N-I-NB+1,

```

```

$          NB, -ONE, WORK( NB+1 ), LDWRKX, A( I, I+NB ), LDA,
$          ONE, A( I+NB, I+NB ), LDA )
*
*      Copy diagonal and off-diagonal elements of B back into A
*
      IF( M.GE.N ) THEN
        DO 10 J = I, I + NB - 1
          A( J, J ) = D( J )
          A( J, J+1 ) = E( J )
10      CONTINUE
        ELSE
          DO 20 J = I, I + NB - 1
            A( J, J ) = D( J )
            A( J+1, J ) = E( J )
20      CONTINUE
        END IF
30 CONTINUE
*
*      Use unblocked code to reduce the remainder of the matrix
*
      CALL DGEBCD2( M-I+1, N-I+1, A( I, I ), LDA, D( I ), E( I ),
$          TAUQ( I ), TAUP( I ), WORK, IINFO )
      WORK( 1 ) = WS
      RETURN
*
*      End of DGEBCD
*
      END

```

— LAPACK dgebrd —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgebrd (m n a lda d e tauq taup work lwork info)
    (declare (type (simple-array double-float (*)) work taup tauq e d a)
              (type fixnum info lwork lda n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (tauq double-float tauq-%data% tauq-%offset%)
       (taup double-float taup-%data% taup-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((ws 0.0) (i 0) (iinfo 0) (j 0) (ldwrkx 0) (ldwrky 0) (lwkopt 0)
              (minmn 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type (double-float) ws)

```

```

      (type fixnum i iinfo j ldwrkx ldwrky lwkopt minmn
        nb nbmin nx)
      (type (member t nil) lquery))
(setf info 0)
(setf nb
  (max (the fixnum 1)
    (the fixnum
      (ilaenv 1 "DGEBRD" " " m n -1 -1))))
(setf lwkopt (f2cl-lib:int-mul (f2cl-lib:int-add m n) nb))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (realpart lwkopt) 'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((< m 0)
    (setf info -1))
  ((< n 0)
    (setf info -2))
  ((< lda (max (the fixnum 1) (the fixnum m)))
    (setf info -4))
  ((and
    (< lwork
      (max (the fixnum 1)
        (the fixnum m)
        (the fixnum n)))
    (not lquery))
    (setf info -10)))
(cond
  ((< info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGEBRD" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(setf minmn (min (the fixnum m) (the fixnum n)))
(cond
  ((= minmn 0)
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum 1) 'double-float))
    (go end_label)))
(setf ws
  (coerce
    (the fixnum
      (max (the fixnum m)
        (the fixnum n)))
    'double-float))
(setf ldwrkx m)
(setf ldwrky n)
(cond
  ((and (> nb 1) (< nb minmn))

```

```

(setf nx
  (max (the fixnum nb)
        (the fixnum
          (ilaenv 3 "DGEBRD" " " m n -1 -1))))
(cond
  ((< nx minmn)
   (setf ws
     (coerce
      (the fixnum
        (f2cl-lib:int-mul (f2cl-lib:int-add m n) nb))
      'double-float)))
  (cond
    ((< lwork ws)
     (setf nbmin (ilaenv 2 "DGEBRD" " " m n -1 -1))
     (cond
      ((>= lwork (f2cl-lib:int-mul (f2cl-lib:int-add m n) nbmin))
       (setf nb (the fixnum (truncate lwork (+ m n)))))
      (t
       (setf nb 1)
       (setf nx minmn)))))))
(t
 (setf nx minmn)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))
  ((> i (f2cl-lib:int-add minmn (f2cl-lib:int-sub nx))) nil)
(tagbody
 (dlabrd (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
  (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) nb
  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
  (f2cl-lib:array-slice d double-float (i) ((1 *)))
  (f2cl-lib:array-slice e double-float (i) ((1 *)))
  (f2cl-lib:array-slice tauq double-float (i) ((1 *)))
  (f2cl-lib:array-slice taup double-float (i) ((1 *))) work ldwrkx
  (f2cl-lib:array-slice work
    double-float
    ((+ (f2cl-lib:int-mul ldwrkx nb) 1))
    ((1 *)))
  ldwrky)
(dgemm "No transpose" "Transpose"
  (f2cl-lib:int-add (f2cl-lib:int-sub m i nb) 1)
  (f2cl-lib:int-add (f2cl-lib:int-sub n i nb) 1) nb (- one)
  (f2cl-lib:array-slice a double-float ((+ i nb) i) ((1 lda) (1 *)))
  lda
  (f2cl-lib:array-slice work
    double-float
    ((+ (f2cl-lib:int-mul ldwrkx nb) nb 1))
    ((1 *)))
  ldwrky one
  (f2cl-lib:array-slice a
    double-float
    ((+ i nb) (f2cl-lib:int-add i nb))

```

```

                                ((1 lda) (1 *)))
lda)
(dgemm "No transpose" "No transpose"
 (f2cl-lib:int-add (f2cl-lib:int-sub m i nb) 1)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i nb) 1) nb (- one)
 (f2cl-lib:array-slice work double-float ((+ nb 1)) ((1 *))) ldwrkx
 (f2cl-lib:array-slice a
  double-float
  (i (f2cl-lib:int-add i nb))
  ((1 lda) (1 *)))
lda one
(f2cl-lib:array-slice a
 double-float
 ((+ i nb) (f2cl-lib:int-add i nb))
 ((1 lda) (1 *)))
lda)
(cond
 ((>= m n)
  (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
   (> j
    (f2cl-lib:int-add i nb (f2cl-lib:int-sub 1)))
   nil)
 (tagbody
  (setf (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)
    (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref a-%data%
    (j (f2cl-lib:int-add j 1))
    ((1 lda) (1 *))
    a-%offset%)
    (f2cl-lib:fref e-%data% (j) ((1 *)) e-%offset%))))))
(t
 (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
  (> j
   (f2cl-lib:int-add i nb (f2cl-lib:int-sub 1)))
  nil)
 (tagbody
  (setf (f2cl-lib:fref a-%data%
    (j j)
    ((1 lda) (1 *))
    a-%offset%)
    (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-add j 1) j)
    ((1 lda) (1 *))
    a-%offset%)
    (f2cl-lib:fref e-%data% (j) ((1 *)) e-%offset%))))))
(multiple-value-bind

```

```

      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dgebd2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
              (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
              (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
              (f2cl-lib:array-slice d double-float (i) ((1 *)))
              (f2cl-lib:array-slice e double-float (i) ((1 *)))
              (f2cl-lib:array-slice tauq double-float (i) ((1 *)))
              (f2cl-lib:array-slice taup double-float (i) ((1 *))) work iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8))
      (setf iinfo var-9))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) ws)
end_label
      (return (values nil nil nil nil nil nil nil nil nil info))))))

```

dgeev LAPACK

— dgeev.input —

```

)set break resume
)sys rm -f dgeev.output
)spool dgeev.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgeev.help —

```

=====
dgeev examples
=====

=====
Man Page Details
=====

```

NAME

DGEEV - for an N-by-N real nonsymmetric matrix A, the eigenvalues and,

optionally, the left and/or right eigenvectors

SYNOPSIS

```
SUBROUTINE DGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR, LDVR,
                  WORK, LWORK, INFO )
```

CHARACTER JOBVL, JOBVR

INTEGER INFO, LDA, LDVL, LDVR, LWORK, N

DOUBLE PRECISION A(LDA, *), VL(LDVL, *), VR(LDVR, *),
 WI(*), WORK(*), WR(*)

PURPOSE

DGEEV computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector $v(j)$ of A satisfies

$$A * v(j) = \lambda(j) * v(j)$$

where $\lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of A satisfies

$$u(j)**H * A = \lambda(j) * u(j)**H$$

where $u(j)**H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

ARGUMENTS

JOBVL (input) CHARACTER*1

= 'N': left eigenvectors of A are not computed;

= 'V': left eigenvectors of A are computed.

JOBVR (input) CHARACTER*1

= 'N': right eigenvectors of A are not computed;

= 'V': right eigenvectors of A are computed.

N (input) INTEGER

The order of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)

On entry, the N-by-N matrix A. On exit, A has been overwritten.

LDA (input) INTEGER

The leading dimension of the array A. $LDA \geq \max(1,N)$.

WR (output) DOUBLE PRECISION array, dimension (N)

WI (output) DOUBLE PRECISION array, dimension (N) WR and
WI contain the real and imaginary parts, respectively, of the

computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

- VL** (output) DOUBLE PRECISION array, dimension (LDVL,N)
 If `JOBVL = 'V'`, the left eigenvectors $u(j)$ are stored one after another in the columns of VL, in the same order as their eigenvalues. If `JOBVL = 'N'`, VL is not referenced. If the j -th eigenvalue is real, then $u(j) = VL(:,j)$, the j -th column of VL. If the j -th and $(j+1)$ -st eigenvalues form a complex conjugate pair, then $u(j) = VL(:,j) + i*VL(:,j+1)$ and $u(j+1) = VL(:,j) - i*VL(:,j+1)$.
- LDVL** (input) INTEGER
 The leading dimension of the array VL. `LDVL` ≥ 1 ; if `JOBVL = 'V'`, `LDVL` $\geq N$.
- VR** (output) DOUBLE PRECISION array, dimension (LDVR,N)
 If `JOBVR = 'V'`, the right eigenvectors $v(j)$ are stored one after another in the columns of VR, in the same order as their eigenvalues. If `JOBVR = 'N'`, VR is not referenced. If the j -th eigenvalue is real, then $v(j) = VR(:,j)$, the j -th column of VR. If the j -th and $(j+1)$ -st eigenvalues form a complex conjugate pair, then $v(j) = VR(:,j) + i*VR(:,j+1)$ and $v(j+1) = VR(:,j) - i*VR(:,j+1)$.
- LDVR** (input) INTEGER
 The leading dimension of the array VR. `LDVR` ≥ 1 ; if `JOBVR = 'V'`, `LDVR` $\geq N$.
- WORK** (workspace/output) DOUBLE PRECISION array, dimension (`MAX(1,LWORK)`)
 On exit, if `INFO = 0`, `WORK(1)` returns the optimal `LWORK`.
- LWORK** (input) INTEGER
 The dimension of the array WORK. `LWORK` $\geq \max(1,3*N)$, and if `JOBVL = 'V'` or `JOBVR = 'V'`, `LWORK` $\geq 4*N$. For good performance, `LWORK` must generally be larger.
- If `LWORK = -1`, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to `LWORK` is issued by XERBLA.
- INFO** (output) INTEGER
 = 0: successful exit
 < 0: if `INFO = -i`, the i -th argument had an illegal value.
 > 0: if `INFO = i`, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements $i+1:N$ of WR and WI contain eigenvalues which have converged.

— dgeev.f —

```

      SUBROUTINE DGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR,
$                      LDVR, WORK, LWORK, INFO )
*
*  -- LAPACK driver routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  December 8, 1999
*
*  .. Scalar Arguments ..
*  CHARACTER          JOBVL, JOBVR
*  INTEGER            INFO, LDA, LDVL, LDVR, LWORK, N
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION   A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
$                    WI( * ), WORK( * ), WR( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
*  DOUBLE PRECISION   ZERO, ONE
*  PARAMETER          ( ZERO = 0.0DO, ONE = 1.0DO )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL            LQUERY, SCALEA, WANTVL, WANTVR
*  CHARACTER          SIDE
*  INTEGER            HSWORK, I, IBAL, IERR, IHI, ILO, ITAU, IWRK, K,
$                    MAXB, MAXWRK, MINWRK, NOUT
*  DOUBLE PRECISION   ANRM, BIGNUM, CS, CSCALE, EPS, R, SCL, SMLNUM,
$                    SN
*
*  ..
*  .. Local Arrays ..
*  LOGICAL            SELECT( 1 )
*  DOUBLE PRECISION   DUM( 1 )
*
*  ..
*  .. External Subroutines ..
*  EXTERNAL           DGEBAK, DGEBAL, DGEHRD, DHSEQR, DLACPY, DLARTG,
$                    DLASCL, DORGHR, DROT, DSCAL, DTREVC, XERBLA
*
*  ..
*  .. External Functions ..
*  LOGICAL            LSAME
*  INTEGER            IDAMAX, ILAENV
*  DOUBLE PRECISION   DLAMCH, DLANGE, DLAPY2, DNRM2

```

```

EXTERNAL          LSAME, IDAMAX, ILAENV, DLAMCH, DLANGE, DLAPY2,
$                 DNRM2
*
* ..
* .. Intrinsic Functions ..
INTRINSIC          MAX, MIN, SQRT
*
* ..
* .. Executable Statements ..
*
* Test the input arguments
*
INFO = 0
LQUERY = ( LWORK.EQ.-1 )
WANTVL = LSAME( JOBVL, 'V' )
WANTVR = LSAME( JOBVR, 'V' )
IF( ( .NOT.WANTVL ) .AND. ( .NOT.LSAME( JOBVL, 'N' ) ) ) THEN
    INFO = -1
ELSE IF( ( .NOT.WANTVR ) .AND. ( .NOT.LSAME( JOBVR, 'N' ) ) ) THEN
    INFO = -2
ELSE IF( N.LT.0 ) THEN
    INFO = -3
ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
    INFO = -5
ELSE IF( LDVL.LT.1 .OR. ( WANTVL .AND. LDVL.LT.N ) ) THEN
    INFO = -9
ELSE IF( LDVR.LT.1 .OR. ( WANTVR .AND. LDVR.LT.N ) ) THEN
    INFO = -11
END IF
*
* Compute workspace
* (Note: Comments in the code beginning "Workspace:" describe the
* minimal amount of workspace needed at that point in the code,
* as well as the preferred amount for good performance.
* NB refers to the optimal block size for the immediately
* following subroutine, as returned by ILAENV.
* HSWORK refers to the workspace preferred by DHSEQR, as
* calculated below. HSWORK is computed assuming ILO=1 and IHI=N,
* the worst case.)
*
MINWRK = 1
IF( INFO.EQ.0 .AND. ( LWORK.GE.1 .OR. LQUERY ) ) THEN
    MAXWRK = 2*N + N*ILAENV( 1, 'DGEHRD', ' ', N, 1, N, 0 )
    IF( ( .NOT.WANTVL ) .AND. ( .NOT.WANTVR ) ) THEN
        MINWRK = MAX( 1, 3*N )
        MAXB = MAX( ILAENV( 8, 'DHSEQR', 'EN', N, 1, N, -1 ), 2 )
        K = MIN( MAXB, N, MAX( 2, ILAENV( 4, 'DHSEQR', 'EN', N, 1,
$           N, -1 ) ) )
        HSWORK = MAX( K*( K+2 ), 2*N )
        MAXWRK = MAX( MAXWRK, N+1, N+HSWORK )
    ELSE
        MINWRK = MAX( 1, 4*N )

```

```

        MAXWRK = MAX( MAXWRK, 2*N+( N-1 )*
$           ILAENV( 1, 'DORGHR', ' ', N, 1, N, -1 ) )
        MAXB = MAX( ILAENV( 8, 'DHSEQR', 'SV', N, 1, N, -1 ), 2 )
        K = MIN( MAXB, N, MAX( 2, ILAENV( 4, 'DHSEQR', 'SV', N, 1,
$           N, -1 ) ) )
        HSWORK = MAX( K*( K+2 ), 2*N )
        MAXWRK = MAX( MAXWRK, N+1, N+HSWORK )
        MAXWRK = MAX( MAXWRK, 4*N )
        END IF
        WORK( 1 ) = MAXWRK
    END IF
    IF( LWORK.LT.MINWRK .AND. .NOT.LQUERY ) THEN
        INFO = -13
    END IF
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DGEEV ', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Quick return if possible
*
    IF( N.EQ.0 )
$   RETURN
*
*   Get machine constants
*
    EPS = DLAMCH( 'P' )
    SMLNUM = DLAMCH( 'S' )
    BIGNUM = ONE / SMLNUM
    CALL DLABAD( SMLNUM, BIGNUM )
    SMLNUM = SQRT( SMLNUM ) / EPS
    BIGNUM = ONE / SMLNUM
*
*   Scale A if max element outside range [SMLNUM,BIGNUM]
*
    ANRM = DLANGE( 'M', N, N, A, LDA, DUM )
    SCALEA = .FALSE.
    IF( ANRM.GT.ZERO .AND. ANRM.LT.SMLNUM ) THEN
        SCALEA = .TRUE.
        CSCALE = SMLNUM
    ELSE IF( ANRM.GT.BIGNUM ) THEN
        SCALEA = .TRUE.
        CSCALE = BIGNUM
    END IF
    IF( SCALEA )
$   CALL DLASCL( 'G', 0, 0, ANRM, CSCALE, N, N, A, LDA, IERR )
*
*   Balance the matrix

```

```

*      (Workspace: need N)
*
      IBAL = 1
      CALL DGEBAL( 'B', N, A, LDA, ILO, IHI, WORK( IBAL ), IERR )
*
*      Reduce to upper Hessenberg form
*      (Workspace: need 3*N, prefer 2*N+N*N)
*
      ITAU = IBAL + N
      IWRK = ITAU + N
      CALL DGEHRD( N, ILO, IHI, A, LDA, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
      IF( WANTVL ) THEN
*
*          Want left eigenvectors
*          Copy Householder vectors to VL
*
          SIDE = 'L'
          CALL DLACPY( 'L', N, N, A, LDA, VL, LDVL )
*
*          Generate orthogonal matrix in VL
*          (Workspace: need 3*N-1, prefer 2*N+(N-1)*N)
*
          CALL DORGHR( N, ILO, IHI, VL, LDVL, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
*          Perform QR iteration, accumulating Schur vectors in VL
*          (Workspace: need N+1, prefer N+HWORK (see comments) )
*
          IWRK = ITAU
          CALL DHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, WR, WI, VL, LDVL,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*
          IF( WANTVR ) THEN
*
*              Want left and right eigenvectors
*              Copy Schur vectors to VR
*
              SIDE = 'B'
              CALL DLACPY( 'F', N, N, VL, LDVL, VR, LDVR )
          END IF
*
      ELSE IF( WANTVR ) THEN
*
*          Want right eigenvectors
*          Copy Householder vectors to VR
*
          SIDE = 'R'
          CALL DLACPY( 'L', N, N, A, LDA, VR, LDVR )

```

```

*
*      Generate orthogonal matrix in VR
*      (Workspace: need 3*N-1, prefer 2*N+(N-1)*NB)
*
      CALL DORGHR( N, ILO, IHI, VR, LDVR, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
*      Perform QR iteration, accumulating Schur vectors in VR
*      (Workspace: need N+1, prefer N+HWORK (see comments) )
*
      IWRK = ITAU
      CALL DHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, WR, WI, VR, LDVR,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*
      ELSE
*
*      Compute eigenvalues only
*      (Workspace: need N+1, prefer N+HWORK (see comments) )
*
      IWRK = ITAU
      CALL DHSEQR( 'E', 'N', N, ILO, IHI, A, LDA, WR, WI, VR, LDVR,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
      END IF
*
*      If INFO > 0 from DHSEQR, then quit
*
      IF( INFO.GT.0 )
$      GO TO 50
*
      IF( WANTVL .OR. WANTVR ) THEN
*
*      Compute left and/or right eigenvectors
*      (Workspace: need 4*N)
*
      CALL DTREVC( SIDE, 'B', SELECT, N, A, LDA, VL, LDVL, VR, LDVR,
$              N, NOUT, WORK( IWRK ), IERR )
      END IF
*
      IF( WANTVL ) THEN
*
*      Undo balancing of left eigenvectors
*      (Workspace: need N)
*
      CALL DGEBAK( 'B', 'L', N, ILO, IHI, WORK( IBAL ), N, VL, LDVL,
$              IERR )
*
*      Normalize left eigenvectors and make largest component real
*
      DO 20 I = 1, N
          IF( WI( I ).EQ.ZERO ) THEN

```

```

        SCL = ONE / DNRM2( N, VL( 1, I ), 1 )
        CALL DSCAL( N, SCL, VL( 1, I ), 1 )
    ELSE IF( WI( I ).GT.ZERO ) THEN
        SCL = ONE / DLAPY2( DNRM2( N, VL( 1, I ), 1 ),
$           DNRM2( N, VL( 1, I+1 ), 1 ) )
        CALL DSCAL( N, SCL, VL( 1, I ), 1 )
        CALL DSCAL( N, SCL, VL( 1, I+1 ), 1 )
        DO 10 K = 1, N
            WORK( IWRK+K-1 ) = VL( K, I )**2 + VL( K, I+1 )**2
10        CONTINUE
        K = IDAMAX( N, WORK( IWRK ), 1 )
        CALL DLARTG( VL( K, I ), VL( K, I+1 ), CS, SN, R )
        CALL DROT( N, VL( 1, I ), 1, VL( 1, I+1 ), 1, CS, SN )
        VL( K, I+1 ) = ZERO
    END IF
20    CONTINUE
    END IF
*
    IF( WANTVR ) THEN
*
*       Undo balancing of right eigenvectors
*       (Workspace: need N)
*
        CALL DGEBAK( 'B', 'R', N, ILO, IHI, WORK( IBAL ), N, VR, LDVR,
$           IERR )
*
*       Normalize right eigenvectors and make largest component real
*
        DO 40 I = 1, N
            IF( WI( I ).EQ.ZERO ) THEN
                SCL = ONE / DNRM2( N, VR( 1, I ), 1 )
                CALL DSCAL( N, SCL, VR( 1, I ), 1 )
            ELSE IF( WI( I ).GT.ZERO ) THEN
                SCL = ONE / DLAPY2( DNRM2( N, VR( 1, I ), 1 ),
$           DNRM2( N, VR( 1, I+1 ), 1 ) )
                CALL DSCAL( N, SCL, VR( 1, I ), 1 )
                CALL DSCAL( N, SCL, VR( 1, I+1 ), 1 )
                DO 30 K = 1, N
                    WORK( IWRK+K-1 ) = VR( K, I )**2 + VR( K, I+1 )**2
30                CONTINUE
                K = IDAMAX( N, WORK( IWRK ), 1 )
                CALL DLARTG( VR( K, I ), VR( K, I+1 ), CS, SN, R )
                CALL DROT( N, VR( 1, I ), 1, VR( 1, I+1 ), 1, CS, SN )
                VR( K, I+1 ) = ZERO
            END IF
40        CONTINUE
    END IF
*
*       Undo scaling if necessary
*

```

```

50 CONTINUE
   IF( SCALEA ) THEN
      CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, N-INFO, 1, WR( INFO+1 ),
$           MAX( N-INFO, 1 ), IERR )
      CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, N-INFO, 1, WI( INFO+1 ),
$           MAX( N-INFO, 1 ), IERR )
      IF( INFO.GT.0 ) THEN
         CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, ILO-1, 1, WR, N,
$           IERR )
         CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, ILO-1, 1, WI, N,
$           IERR )
      END IF
   END IF
   RETURN
*
*   WORK( 1 ) = MAXWRK
*   RETURN
*
*   End of DGEEV
*
END

```

— LAPACK dgeev —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgeev (jobvl jobvr n a lda wr wi vl ldvl vr ldvr work lwork info)
    (declare (type (simple-array double-float (*)) work vr vl wi wr a)
              (type fixnum info lwork ldvr ldvl lda n)
              (type character jobvr jobvl))
    (f2cl-lib:with-multi-array-data
      ((jobvl character jobvl-%data% jobvl-%offset%)
       (jobvr character jobvr-%data% jobvr-%offset%)
       (a double-float a-%data% a-%offset%)
       (wr double-float wr-%data% wr-%offset%)
       (wi double-float wi-%data% wi-%offset%)
       (vl double-float vl-%data% vl-%offset%)
       (vr double-float vr-%data% vr-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float))
              (select (make-array 1 :element-type 't)) (anrm 0.0) (bignum 0.0)
              (cs 0.0) (cscale 0.0) (eps 0.0) (r 0.0) (scl 0.0) (smlnum 0.0)
              (sn 0.0) (hswrk 0) (i 0) (ibal 0) (ierr 0) (ihi 0) (ilo 0)
              (itau 0) (iwrk 0) (k 0) (maxb 0) (maxwrk 0) (minwrk 0) (nout 0)
              (side
                (make-array '(1) :element-type 'character :initial-element #\ )

```

```

(lquery nil) (scalea nil) (wantvl nil) (wantvr nil))
(declare (type (simple-array double-float (1)) dum)
  (type (simple-array (member t nil) (1)) select)
  (type (double-float) anrm bignum cs cscale eps r scl smlnum
        sn)
  (type fixnum hswork i ibal ierr ihi ilo itau iwrk
        k maxb maxwrk minwrk nout)
  (type (simple-array character (1)) side)
  (type (member t nil) lquery scalea wantvl wantvr))
(setf info 0)
(setf lquery (coerce (= lwork -1) '(member t nil)))
(setf wantvl (char-equal jobvl #\V))
(setf wantvr (char-equal jobvr #\V))
(cond
  ((and (not wantvl) (not (char-equal jobvl #\N)))
   (setf info -1))
  ((and (not wantvr) (not (char-equal jobvr #\N)))
   (setf info -2))
  ((< n 0)
   (setf info -3))
  ((< lda (max (the fixnum 1) (the fixnum n)))
   (setf info -5))
  ((or (< ldvl 1) (and wantvl (< ldvl n)))
   (setf info -9))
  ((or (< ldvr 1) (and wantvr (< ldvr n)))
   (setf info -11)))
(setf minwrk 1)
(cond
  ((and (= info 0) (or (>= lwork 1) lquery))
   (setf maxwrk
     (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
                       (f2cl-lib:int-mul n
                                           (ilaenv 1 "DGEHRD" " " " n
                                                    1 n 0))))))
(cond
  ((and (not wantvl) (not wantvr))
   (setf minwrk
     (max (the fixnum 1)
           (the fixnum (f2cl-lib:int-mul 3 n))))
   (setf maxb
     (max
      (the fixnum
        (ilaenv 8 "DHSEQR" "EN" n 1 n -1))
      (the fixnum 2)))
   (setf k
     (min (the fixnum maxb)
           (the fixnum n)
           (the fixnum
            (max (the fixnum 2)
                  (the fixnum
                   (ilaenv 1 "DGEHRD" " " " n
                          1 n 0)))))))

```



```

                                (ilaenv 4 "DHSEQR" "EN" n 1 n -1))))))
  (setf hswork
    (max
      (the fixnum
        (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
      (the fixnum (f2cl-lib:int-mul 2 n))))
  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum (f2cl-lib:int-add n 1))
      (the fixnum
        (f2cl-lib:int-add n hswork)))))
  (t
    (setf minwrk
      (max (the fixnum 1)
        (the fixnum (f2cl-lib:int-mul 4 n))))
    (setf maxwrk
      (max (the fixnum maxwrk)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
            (f2cl-lib:int-mul
              (f2cl-lib:int-sub n 1)
              (ilaenv 1 "DORGHR" " " n 1 n
                -1)))))))
    (setf maxb
      (max
        (the fixnum
          (ilaenv 8 "DHSEQR" "SV" n 1 n -1))
        (the fixnum 2)))
    (setf k
      (min (the fixnum maxb)
        (the fixnum n)
        (the fixnum
          (max (the fixnum 2)
            (the fixnum
              (ilaenv 4 "DHSEQR" "SV" n 1 n -1)))))))
    (setf hswork
      (max
        (the fixnum
          (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
        (the fixnum (f2cl-lib:int-mul 2 n))))
    (setf maxwrk
      (max (the fixnum maxwrk)
        (the fixnum (f2cl-lib:int-add n 1))
        (the fixnum (f2cl-lib:int-add n hswork))))
    (setf maxwrk
      (max (the fixnum maxwrk)
        (the fixnum (f2cl-lib:int-mul 4 n))))))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (coerce (the fixnum maxwrk) 'double-float)))
(cond

```

```

      ((and (< lwork minwrk) (not lquery))
       (setf info -13)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEEV " (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(if (= n 0) (go end_label))
(setf eps (dlamch "P"))
(setf smlnum (dlamch "S"))
(setf bignum (/ one smlnum))
(multiple-value-bind (var-0 var-1)
  (dlabad smlnum bignum)
  (declare (ignore))
  (setf smlnum var-0)
  (setf bignum var-1))
(setf smlnum (/ (f2cl-lib:fsqrt smlnum) eps))
(setf bignum (/ one smlnum))
(setf anrm (dlange "M" n n a lda dum))
(setf scalea nil)
(cond
  ((and (> anrm zero) (< anrm smlnum))
   (setf scalea t)
   (setf cscale smlnum))
  ((> anrm bignum)
   (setf scalea t)
   (setf cscale bignum)))
(if scalea
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (dlascl "G" 0 0 anrm cscale n n a lda ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                     var-8))
    (setf ierr var-9)))
  (setf ibal 1)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgebal "B" n a lda ilo ihi
      (f2cl-lib:array-slice work double-float (ibal) ((1 *))) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-6))
    (setf ilo var-4)
    (setf ihi var-5)
    (setf ierr var-7))
  (setf itau (f2cl-lib:int-add ibal n))
  (setf iwrk (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dgehrd n ilo ihi a lda

```

```

      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
(setf ierr var-8))
(cond
 (wantvl
  (f2cl-lib:f2cl-set-string side "L" (string 1))
  (dlacpy "L" n n a lda vl ldvl)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
   (dorghr n ilo ihi vl ldvl
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
   (setf ierr var-8))
  (setf iwrk itau)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13)
   (dhseqr "S" "V" n ilo ihi a lda wr wi vl ldvl
    (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10 var-11 var-12))
   (setf info var-13))
  (cond
   (wantvr
    (f2cl-lib:f2cl-set-string side "B" (string 1))
    (dlacpy "F" n n vl ldvl vr ldvr))))
 (wantvr
  (f2cl-lib:f2cl-set-string side "R" (string 1))
  (dlacpy "L" n n a lda vr ldvr)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
   (dorghr n ilo ihi vr ldvr
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
   (setf ierr var-8))
  (setf iwrk itau)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13)
   (dhseqr "S" "V" n ilo ihi a lda wr wi vr ldvr
    (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```

```

                                var-8 var-9 var-10 var-11 var-12))
    (setf info var-13)))
  (t
   (setf iwrk itau)
   (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
     var-10 var-11 var-12 var-13)
    (dhseqr "E" "N" n ilo ihi a lda wr wi vr ldvr
     (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))))
  (if (> info 0) (go label50))
  (cond
   ((or wantvl wantvr)
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
      var-10 var-11 var-12 var-13)
     (dtrevc side "B" select n a lda vl ldvl vr ldvr n nout
      (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10 var-12))
     (setf nout var-11)
     (setf ierr var-13))))
   (cond
    (wantvl
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dgebak "B" "L" n ilo ihi
       (f2cl-lib:array-slice work double-float (ibal) ((1 *))) n vl
       ldvl ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8))
      (setf ierr var-9))
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i n) nil)
     (tagbody
      (cond
       ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
        (setf scl
         (/ one
          (dnrm2 n
           (f2cl-lib:array-slice vl
            double-float
            (1 i)
            ((1 ldvl) (1 *)))
           1))))
      (dscal n scl
       (f2cl-lib:array-slice vl

```

```

                                double-float
                                (1 i)
                                ((1 ldvl) (1 *)))
1))
(> (f2cl-lib:fref wi (i) ((1 *))) zero)
(setf scl
  (/ one
    (dlapy2
      (dnrm2 n
        (f2cl-lib:array-slice vl
          double-float
          (1 i)
          ((1 ldvl) (1 *)))
        1)
      (dnrm2 n
        (f2cl-lib:array-slice vl
          double-float
          (1 (f2cl-lib:int-add i 1))
          ((1 ldvl) (1 *)))
        1))))
(dscal n scl
  (f2cl-lib:array-slice vl
    double-float
    (1 i)
    ((1 ldvl) (1 *)))
  1)
(dscal n scl
  (f2cl-lib:array-slice vl
    double-float
    (1 (f2cl-lib:int-add i 1))
    ((1 ldvl) (1 *)))
  1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-sub
        (f2cl-lib:int-add iwrk k)
        1))
      ((1 *))
      work-%offset%))
      (+
        (expt
          (f2cl-lib:fref vl-%data%
            (k i)
            ((1 ldvl) (1 *))
            vl-%offset%))
          2)
        (expt
          (f2cl-lib:fref vl-%data%

```

```

(k (f2cl-lib:int-add i 1))
((1 ldvl) (1 *))
vl-%offset%)
2))))))
(setf k
  (idamax n
    (f2cl-lib:array-slice work
      double-float
      (iwrk)
      ((1 *)))
    1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg
    (f2cl-lib:fref vl-%data%
      (k i)
      ((1 ldvl) (1 *))
      vl-%offset%)
    (f2cl-lib:fref vl-%data%
      (k (f2cl-lib:int-add i 1))
      ((1 ldvl) (1 *))
      vl-%offset%)
    cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(drot n
  (f2cl-lib:array-slice vl
    double-float
    (1 i)
    ((1 ldvl) (1 *)))
  1
  (f2cl-lib:array-slice vl
    double-float
    (1 (f2cl-lib:int-add i 1))
    ((1 ldvl) (1 *)))
  1 cs sn)
(setf (f2cl-lib:fref vl-%data%
  (k (f2cl-lib:int-add i 1))
  ((1 ldvl) (1 *))
  vl-%offset%)
  zero))))))
(cond
  (wantvr
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dgebak "B" "R" n ilo ihi
        (f2cl-lib:array-slice work double-float (ibal) ((1 *))) n vr
        ldvr ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```

```

                                var-8))
  (setf ierr var-9))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (cond
      ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
        (setf scl
          (/ one
            (dnrm2 n
              (f2cl-lib:array-slice vr
                double-float
                (1 i)
                ((1 ldvr) (1 *)))
              1)))
        (dscal n scl
          (f2cl-lib:array-slice vr
            double-float
            (1 i)
            ((1 ldvr) (1 *)))
          1))
      (> (f2cl-lib:fref wi (i) ((1 *))) zero)
        (setf scl
          (/ one
            (dlapy2
              (dnrm2 n
                (f2cl-lib:array-slice vr
                  double-float
                  (1 i)
                  ((1 ldvr) (1 *)))
                1)
              (dnrm2 n
                (f2cl-lib:array-slice vr
                  double-float
                  (1 (f2cl-lib:int-add i 1))
                  ((1 ldvr) (1 *)))
                1))))
        (dscal n scl
          (f2cl-lib:array-slice vr
            double-float
            (1 i)
            ((1 ldvr) (1 *)))
          1)
        (dscal n scl
          (f2cl-lib:array-slice vr
            double-float
            (1 (f2cl-lib:int-add i 1))
            ((1 ldvr) (1 *)))
          1)
        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))

```

```

        (> k n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-sub
                        (f2cl-lib:int-add iwrk k)
                        1))
                      ((1 *))
                      work-%offset%)
    (+
      (expt
        (f2cl-lib:fref vr-%data%
                      (k i)
                      ((1 ldvr) (1 *))
                      vr-%offset%)
        2)
      (expt
        (f2cl-lib:fref vr-%data%
                      (k (f2cl-lib:int-add i 1))
                      ((1 ldvr) (1 *))
                      vr-%offset%)
        2))))))
(setf k
  (idamax n
    (f2cl-lib:array-slice work
                          double-float
                          (iwrk)
                          ((1 *)))
    1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg
    (f2cl-lib:fref vr-%data%
                  (k i)
                  ((1 ldvr) (1 *))
                  vr-%offset%)
    (f2cl-lib:fref vr-%data%
                  (k (f2cl-lib:int-add i 1))
                  ((1 ldvr) (1 *))
                  vr-%offset%)
    cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(drot n
  (f2cl-lib:array-slice vr
                        double-float
                        (1 i)
                        ((1 ldvr) (1 *)))
  1
  (f2cl-lib:array-slice vr

```



```

                                double-float
                                (1 (f2cl-lib:int-add i 1))
                                ((1 ldvr) (1 *)))

      1 cs sn)
      (setf (f2cl-lib:fref vr-%data%
                            (k (f2cl-lib:int-add i 1))
                            ((1 ldvr) (1 *)))
            vr-%offset%)

      zero))))))

label50
  (cond
    (scalea
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
        (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
          (f2cl-lib:array-slice wr double-float ((+ info 1)) ((1 *)))
          (max (the fixnum (f2cl-lib:int-sub n info))
              (the fixnum 1))
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8))
        (setf ierr var-9))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
        (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
          (f2cl-lib:array-slice wi double-float ((+ info 1)) ((1 *)))
          (max (the fixnum (f2cl-lib:int-sub n info))
              (the fixnum 1))
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8))
        (setf ierr var-9))
      (cond
        ((> info 0)
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
              var-9)
            (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wr n
              ierr)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
              var-7 var-8))
            (setf ierr var-9))
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
              var-9)
            (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wi n
              ierr)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
              var-7 var-8))
            (setf ierr var-9))))))

```

```

      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
            (coerce (the fixnum maxwrk) 'double-float))
end_label
(return
 (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dgeevx LAPACK

— dgeevx.input —

```

)set break resume
)sys rm -f dgeevx.output
)spool dgeevx.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgeevx.help —

```
=====
dgeevx examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGEEVX - for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors

SYNOPSIS

```

SUBROUTINE DGEEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI, VL,
                  LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE,
                  RCONDV, WORK, LWORK, IWORK, INFO )

```

CHARACTER BALANC, JOBVL, JOBVR, SENSE

```

INTEGER      IHI, ILO, INFO, LDA, LDVL, LDVR, LWORK, N

DOUBLE       PRECISION ABNRM

INTEGER      IWORK( * )

DOUBLE       PRECISION A( LDA, * ), RCONDE( * ), RCONDV( * ),
              SCALE( * ), VL( LDVL, * ), VR( LDVR, * ), WI( * ),
              WORK( * ), WR( * )

```

PURPOSE

DGEEVX computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector $v(j)$ of A satisfies

$$A * v(j) = \text{lambda}(j) * v(j)$$

where $\text{lambda}(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of A satisfies

$$u(j)**H * A = \text{lambda}(j) * u(j)**H$$

where $u(j)**H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation $D * A * D^{**}(-1)$, where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.10.2 of the LAPACK Users' Guide.

ARGUMENTS

BALANC (input) CHARACTER*1

Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues.
 = 'N': Do not diagonally scale or permute;

= 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale; = 'S': Diagonally scale the matrix, i.e. replace A by $D*A*D^{**}(-1)$, where D is a diagonal matrix chosen to make the rows and columns of A more

equal in norm. Do not permute; = 'B': Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

- JOBVL** (input) CHARACTER*1
 = 'N': left eigenvectors of A are not computed;
 = 'V': left eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVL must = 'V'.
- JOBVR** (input) CHARACTER*1
 = 'N': right eigenvectors of A are not computed;
 = 'V': right eigenvectors of A are computed. If SENSE = 'E' or 'B', JOBVR must = 'V'.
- SENSE** (input) CHARACTER*1
 Determines which reciprocal condition numbers are computed. =
 'N': None are computed;
 = 'E': Computed for eigenvalues only;
 = 'V': Computed for right eigenvectors only;
 = 'B': Computed for eigenvalues and right eigenvectors.
- If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').
- N** (input) INTEGER
 The order of the matrix A. $N \geq 0$.
- A** (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the N-by-N matrix A. On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the real Schur form of the balanced version of the input matrix A.
- LDA** (input) INTEGER
 The leading dimension of the array A. $LDA \geq \max(1,N)$.
- WR** (output) DOUBLE PRECISION array, dimension (N)
WI (output) DOUBLE PRECISION array, dimension (N) WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
- VL** (output) DOUBLE PRECISION array, dimension (LDVL,N)
 If JOBVL = 'V', the left eigenvectors $u(j)$ are stored one after another in the columns of VL, in the same order as their eigenvalues. If JOBVL = 'N', VL is not referenced. If the j-th eigenvalue is real, then $u(j) = VL(:,j)$, the j-th column of VL.

If the j -th and $(j+1)$ -st eigenvalues form a complex conjugate pair, then $u(j) = VL(:,j) + i*VL(:,j+1)$ and $u(j+1) = VL(:,j) - i*VL(:,j+1)$.

- LDVL** (input) INTEGER
The leading dimension of the array VL. LDVL ≥ 1 ; if JOBVL = 'V', LDVL $\geq N$.
- VR** (output) DOUBLE PRECISION array, dimension (LDVR,N)
If JOBVR = 'V', the right eigenvectors $v(j)$ are stored one after another in the columns of VR, in the same order as their eigenvalues. If JOBVR = 'N', VR is not referenced. If the j -th eigenvalue is real, then $v(j) = VR(:,j)$, the j -th column of VR. If the j -th and $(j+1)$ -st eigenvalues form a complex conjugate pair, then $v(j) = VR(:,j) + i*VR(:,j+1)$ and $v(j+1) = VR(:,j) - i*VR(:,j+1)$.
- LDVR** (input) INTEGER
The leading dimension of the array VR. LDVR ≥ 1 , and if JOBVR = 'V', LDVR $\geq N$.
- ILO** (output) INTEGER
IHI (output) INTEGER ILO and IHI are integer values determined when A was balanced. The balanced $A(i,j) = 0$ if $I > J$ and $J = 1, \dots, ILO-1$ or $I = IHI+1, \dots, N$.
- SCALE** (output) DOUBLE PRECISION array, dimension (N)
Details of the permutations and scaling factors applied when balancing A. If $P(j)$ is the index of the row and column interchanged with row and column j , and $D(j)$ is the scaling factor applied to row and column j , then $SCALE(J) = P(J)$, for $J = 1, \dots, ILO-1$; $SCALE(J) = D(J)$, for $J = ILO, \dots, IHI$; $SCALE(J) = P(J)$, for $J = IHI+1, \dots, N$. The order in which the interchanges are made is N to IHI+1, then 1 to ILO-1.
- ABNRM** (output) DOUBLE PRECISION
The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).
- RCONDE** (output) DOUBLE PRECISION array, dimension (N)
 $RCONDE(j)$ is the reciprocal condition number of the j -th eigenvalue.
- RCONDV** (output) DOUBLE PRECISION array, dimension (N)
 $RCONDV(j)$ is the reciprocal condition number of the j -th right eigenvector.
- WORK** (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
 The dimension of the array WORK. If SENSE = 'N' or 'E', LWORK $\geq \max(1, 2*N)$, and if JOBVL = 'V' or JOBVR = 'V', LWORK $\geq 3*N$. If SENSE = 'V' or 'B', LWORK $\geq N*(N+6)$. For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace) INTEGER array, dimension (2*N-2)
 If SENSE = 'N' or 'E', not referenced.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ILO-1 and i+1:N of WR and WI contain eigenvalues which have converged.

— dgeevx.f —

```

SUBROUTINE DGEEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI,
$                 VL, LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM,
$                 RCONDE, RCONDV, WORK, LWORK, IWORK, INFO )
*
*  -- LAPACK driver routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
CHARACTER          BALANC, JOBVL, JOBVR, SENSE
INTEGER            IHI, ILO, INFO, LDA, LDVL, LDVR, LWORK, N
DOUBLE PRECISION   ABNRM
*
*  ..
*  .. Array Arguments ..
INTEGER            IWORK( * )
DOUBLE PRECISION   A( LDA, * ), RCONDE( * ), RCONDV( * ),
$                 SCALE( * ), VL( LDVL, * ), VR( LDVR, * ),
$                 WI( * ), WORK( * ), WR( * )
*
*  ..
*

```

```

* =====
*
* .. Parameters ..
  DOUBLE PRECISION  ZERO, ONE
  PARAMETER          ( ZERO = 0.0DO, ONE = 1.0DO )
*
* ..
* .. Local Scalars ..
  LOGICAL            LQUERY, SCALEA, WANTVL, WANTVR, WNTSNB, WNTSNE,
$                   WNTSNN, WNTSNV
  CHARACTER          JOB, SIDE
  INTEGER            HSWORK, I, ICOND, IERR, ITAU, IWRK, K, MAXB,
$                   MAXWRK, MINWRK, NOUT
  DOUBLE PRECISION  ANRM, BIGNUM, CS, CSCALE, EPS, R, SCL, SMLNUM,
$                   SN
*
* ..
* .. Local Arrays ..
  LOGICAL            SELECT( 1 )
  DOUBLE PRECISION  DUM( 1 )
*
* ..
* .. External Subroutines ..
  EXTERNAL           DGEBAK, DGEBAL, DGEHRD, DHSEQR, DLACPY, DLARTG,
$                   DLASCL, DORGHR, DROT, DSCAL, DTREVC, DTRSNA,
$                   XERBLA
*
* ..
* .. External Functions ..
  LOGICAL            LSAME
  INTEGER            IDAMAX, ILAENV
  DOUBLE PRECISION  DLAMCH, DLANGE, DLAPY2, DNRM2
  EXTERNAL           LSAME, IDAMAX, ILAENV, DLAMCH, DLANGE, DLAPY2,
$                   DNRM2
*
* ..
* .. Intrinsic Functions ..
  INTRINSIC          MAX, MIN, SQRT
*
* ..
* .. Executable Statements ..
*
* Test the input arguments
*
  INFO = 0
  LQUERY = ( LWORK.EQ.-1 )
  WANTVL = LSAME( JOBVL, 'V' )
  WANTVR = LSAME( JOBVR, 'V' )
  WNTSNN = LSAME( SENSE, 'N' )
  WNTSNE = LSAME( SENSE, 'E' )
  WNTSNV = LSAME( SENSE, 'V' )
  WNTSNB = LSAME( SENSE, 'B' )
  IF( .NOT.( LSAME( BALANC, 'N' ) .OR. LSAME( BALANC,
$   'S' ) .OR. LSAME( BALANC, 'P' ) .OR. LSAME( BALANC, 'B' ) ) )
$   THEN
    INFO = -1

```

```

ELSE IF( ( .NOT.WANTVL ) .AND. ( .NOT.LSAME( JOBVL, 'N' ) ) ) THEN
    INFO = -2
ELSE IF( ( .NOT.WANTVR ) .AND. ( .NOT.LSAME( JOBVR, 'N' ) ) ) THEN
    INFO = -3
ELSE IF( .NOT.( WNTSNN .OR. WNTSNE .OR. WNTSNB .OR. WNTSNV ) .OR.
$      ( ( WNTSNE .OR. WNTSNB ) .AND. .NOT.( WANTVL .AND.
$      WANTVR ) ) ) THEN
    INFO = -4
ELSE IF( N.LT.0 ) THEN
    INFO = -5
ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
    INFO = -7
ELSE IF( LDVL.LT.1 .OR. ( WANTVL .AND. LDVL.LT.N ) ) THEN
    INFO = -11
ELSE IF( LDVR.LT.1 .OR. ( WANTVR .AND. LDVR.LT.N ) ) THEN
    INFO = -13
END IF

*
*   Compute workspace
*   (Note: Comments in the code beginning "Workspace:" describe the
*   minimal amount of workspace needed at that point in the code,
*   as well as the preferred amount for good performance.
*   NB refers to the optimal block size for the immediately
*   following subroutine, as returned by ILAENV.
*   HSWORK refers to the workspace preferred by DHSEQR, as
*   calculated below. HSWORK is computed assuming ILO=1 and IHI=N,
*   the worst case.)
*

MINWRK = 1
IF( INFO.EQ.0 .AND. ( LWORK.GE.1 .OR. LQUERY ) ) THEN
    MAXWRK = N + N*ILAENV( 1, 'DGEHRD', ' ', N, 1, N, 0 )
    IF( ( .NOT.WANTVL ) .AND. ( .NOT.WANTVR ) ) THEN
        MINWRK = MAX( 1, 2*N )
        IF( .NOT.WNTSNN )
$      MINWRK = MAX( MINWRK, N*N+6*N )
        MAXB = MAX( ILAENV( 8, 'DHSEQR', 'SN', N, 1, N, -1 ), 2 )
        IF( WNTSNN ) THEN
            K = MIN( MAXB, N, MAX( 2, ILAENV( 4, 'DHSEQR', 'EN', N,
$      1, N, -1 ) ) )
        ELSE
            K = MIN( MAXB, N, MAX( 2, ILAENV( 4, 'DHSEQR', 'SN', N,
$      1, N, -1 ) ) )
        END IF
        HSWORK = MAX( K*( K+2 ), 2*N )
        MAXWRK = MAX( MAXWRK, 1, HSWORK )
        IF( .NOT.WNTSNN )
$      MAXWRK = MAX( MAXWRK, N*N+6*N )
    ELSE
        MINWRK = MAX( 1, 3*N )
        IF( ( .NOT.WNTSNN ) .AND. ( .NOT.WNTSNE ) )

```



```

$      MINWRK = MAX( MINWRK, N*N+6*N )
      MAXB = MAX( ILAENV( 8, 'DHSEQR', 'SN', N, 1, N, -1 ), 2 )
      K = MIN( MAXB, N, MAX( 2, ILAENV( 4, 'DHSEQR', 'EN', N, 1,
$      N, -1 ) ) )
      HSWORK = MAX( K*( K+2 ), 2*N )
      MAXWRK = MAX( MAXWRK, 1, HSWORK )
      MAXWRK = MAX( MAXWRK, N*( N-1 ) *
$      ILAENV( 1, 'DORGHR', ' ', N, 1, N, -1 ) )
      IF( ( .NOT.WNTSNN ) .AND. ( .NOT.WNTSNE ) )
$      MAXWRK = MAX( MAXWRK, N*N+6*N )
      MAXWRK = MAX( MAXWRK, 3*N, 1 )
      END IF
      WORK( 1 ) = MAXWRK
      END IF
      IF( LWORK.LT.MINWRK .AND. .NOT.LQUERY ) THEN
          INFO = -21
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DGEEVX', -INFO )
          RETURN
      ELSE IF( LQUERY ) THEN
          RETURN
      END IF

*
*      Quick return if possible
*
      IF( N.EQ.0 )
$      RETURN

*
*      Get machine constants
*
      EPS = DLAMCH( 'P' )
      SMLNUM = DLAMCH( 'S' )
      BIGNUM = ONE / SMLNUM
      CALL DLABAD( SMLNUM, BIGNUM )
      SMLNUM = SQRT( SMLNUM ) / EPS
      BIGNUM = ONE / SMLNUM

*
*      Scale A if max element outside range [SMLNUM,BIGNUM]
*
      ICOND = 0
      ANRM = DLANGE( 'M', N, N, A, LDA, DUM )
      SCALEA = .FALSE.
      IF( ANRM.GT.ZERO .AND. ANRM.LT.SMLNUM ) THEN
          SCALEA = .TRUE.
          CSCALE = SMLNUM
      ELSE IF( ANRM.GT.BIGNUM ) THEN
          SCALEA = .TRUE.
          CSCALE = BIGNUM
      END IF

```

```

      IF( SCALEA )
$     CALL DLASCL( 'G', 0, 0, ANRM, CSCALE, N, N, A, LDA, IERR )
*
*     Balance the matrix and compute ABNRM
*
      CALL DGEBAL( BALANC, N, A, LDA, ILO, IHI, SCALE, IERR )
      ABNRM = DLANGE( '1', N, N, A, LDA, DUM )
      IF( SCALEA ) THEN
          DUM( 1 ) = ABNRM
          CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, 1, 1, DUM, 1, IERR )
          ABNRM = DUM( 1 )
      END IF
*
*     Reduce to upper Hessenberg form
*     (Workspace: need 2*N, prefer N+N*NB)
*
      ITAU = 1
      IWRK = ITAU + N
      CALL DGEHRD( N, ILO, IHI, A, LDA, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
      IF( WANTVL ) THEN
*
*         Want left eigenvectors
*         Copy Householder vectors to VL
*
          SIDE = 'L'
          CALL DLACPY( 'L', N, N, A, LDA, VL, LDVL )
*
*         Generate orthogonal matrix in VL
*         (Workspace: need 2*N-1, prefer N+(N-1)*NB)
*
          CALL DORGHR( N, ILO, IHI, VL, LDVL, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
*         Perform QR iteration, accumulating Schur vectors in VL
*         (Workspace: need 1, prefer HSWORK (see comments) )
*
          IWRK = ITAU
          CALL DHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, WR, WI, VL, LDVL,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*
      IF( WANTVR ) THEN
*
*         Want left and right eigenvectors
*         Copy Schur vectors to VR
*
          SIDE = 'B'
          CALL DLACPY( 'F', N, N, VL, LDVL, VR, LDVR )
      END IF

```

```

*
*      ELSE IF( WANTVR ) THEN
*
*          Want right eigenvectors
*          Copy Householder vectors to VR
*
*          SIDE = 'R'
*          CALL DLACPY( 'L', N, N, A, LDA, VR, LDVR )
*
*          Generate orthogonal matrix in VR
*          (Workspace: need 2*N-1, prefer N+(N-1)*NB)
*
*          CALL DORGHR( N, ILO, IHI, VR, LDVR, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
*          Perform QR iteration, accumulating Schur vectors in VR
*          (Workspace: need 1, prefer HSWORK (see comments) )
*
*          IWRK = ITAU
*          CALL DHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, WR, WI, VR, LDVR,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*
*      ELSE
*
*          Compute eigenvalues only
*          If condition numbers desired, compute Schur form
*
*          IF( WNTSNN ) THEN
*              JOB = 'E'
*          ELSE
*              JOB = 'S'
*          END IF
*
*          (Workspace: need 1, prefer HSWORK (see comments) )
*
*          IWRK = ITAU
*          CALL DHSEQR( JOB, 'N', N, ILO, IHI, A, LDA, WR, WI, VR, LDVR,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*          END IF
*
*          If INFO > 0 from DHSEQR, then quit
*
*          IF( INFO.GT.0 )
$              GO TO 50
*
*          IF( WANTVL .OR. WANTVR ) THEN
*
*              Compute left and/or right eigenvectors
*              (Workspace: need 3*N)

```

```

      CALL DTREVC( SIDE, 'B', SELECT, N, A, LDA, VL, LDVL, VR, LDVR,
$           N, NOUT, WORK( IWRK ), IERR )
      END IF

*
*   Compute condition numbers if desired
*   (Workspace: need N*N+6*N unless SENSE = 'E')
*
      IF( .NOT.WNTSNN ) THEN
          CALL DTRSNA( SENSE, 'A', SELECT, N, A, LDA, VL, LDVL, VR, LDVR,
$               RCONDE, RCONDV, N, NOUT, WORK( IWRK ), N, IWORK,
$               ICOND )
      END IF

*
      IF( WANTVL ) THEN

*           Undo balancing of left eigenvectors
*
          CALL DGEBAK( BALANC, 'L', N, ILO, IHI, SCALE, N, VL, LDVL,
$               IERR )

*           Normalize left eigenvectors and make largest component real
*
          DO 20 I = 1, N
              IF( WI( I ).EQ.ZERO ) THEN
                  SCL = ONE / DNRM2( N, VL( 1, I ), 1 )
                  CALL DSCAL( N, SCL, VL( 1, I ), 1 )
              ELSE IF( WI( I ).GT.ZERO ) THEN
                  SCL = ONE / DLAPY2( DNRM2( N, VL( 1, I ), 1 ),
$                      DNRM2( N, VL( 1, I+1 ), 1 ) )
                  CALL DSCAL( N, SCL, VL( 1, I ), 1 )
                  CALL DSCAL( N, SCL, VL( 1, I+1 ), 1 )
                  DO 10 K = 1, N
                      WORK( K ) = VL( K, I )**2 + VL( K, I+1 )**2
10                  CONTINUE
                  K = IDAMAX( N, WORK, 1 )
                  CALL DLARTG( VL( K, I ), VL( K, I+1 ), CS, SN, R )
                  CALL DROT( N, VL( 1, I ), 1, VL( 1, I+1 ), 1, CS, SN )
                  VL( K, I+1 ) = ZERO
              END IF
          20 CONTINUE
      END IF

*
      IF( WANTVR ) THEN

*           Undo balancing of right eigenvectors
*
          CALL DGEBAK( BALANC, 'R', N, ILO, IHI, SCALE, N, VR, LDVR,
$               IERR )

*           Normalize right eigenvectors and make largest component real

```

```

*
      DO 40 I = 1, N
        IF( WI( I ).EQ.ZERO ) THEN
          SCL = ONE / DNRM2( N, VR( 1, I ), 1 )
          CALL DSCAL( N, SCL, VR( 1, I ), 1 )
        ELSE IF( WI( I ).GT.ZERO ) THEN
          SCL = ONE / DLAPY2( DNRM2( N, VR( 1, I ), 1 ),
$           DNRM2( N, VR( 1, I+1 ), 1 ) )
          CALL DSCAL( N, SCL, VR( 1, I ), 1 )
          CALL DSCAL( N, SCL, VR( 1, I+1 ), 1 )
          DO 30 K = 1, N
            WORK( K ) = VR( K, I )**2 + VR( K, I+1 )**2
30          CONTINUE
            K = IDAMAX( N, WORK, 1 )
            CALL DLARTG( VR( K, I ), VR( K, I+1 ), CS, SN, R )
            CALL DROT( N, VR( 1, I ), 1, VR( 1, I+1 ), 1, CS, SN )
            VR( K, I+1 ) = ZERO
          END IF
        40 CONTINUE
      END IF
*
*      Undo scaling if necessary
*
50 CONTINUE
  IF( SCALEA ) THEN
    CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, N-INFO, 1, VR( INFO+1 ),
$     MAX( N-INFO, 1 ), IERR )
    CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, N-INFO, 1, WI( INFO+1 ),
$     MAX( N-INFO, 1 ), IERR )
    IF( INFO.EQ.0 ) THEN
      IF( ( WNTSNV .OR. WNTSNB ) .AND. ICOND.EQ.0 )
$       CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, N, 1, RCONDV, N,
$       IERR )
    ELSE
      CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, ILO-1, 1, VR, N,
$       IERR )
      CALL DLASCL( 'G', 0, 0, CSCALE, ANRM, ILO-1, 1, WI, N,
$       IERR )
    END IF
  END IF
*
  WORK( 1 ) = MAXWRK
  RETURN
*
*      End of DGEEVX
*
END

```

— LAPACK dgeevx —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgeevx
    (balanc jobvl jobvr sense n a lda wr wi vl ldvl vr ldvr ilo ihi scale
     abnrm rconde rcondv work lwork iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
              (type (double-float) abnrm)
              (type (simple-array double-float (*)) work rcondv rconde scale vr vl wi
                    wr a)
              (type fixnum info lwork ihi ilo ldvr ldvl lda n)
              (type character sense jobvr jobvl balanc))
    (f2cl-lib:with-multi-array-data
      ((balanc character balanc-%data% balanc-%offset%)
       (jobvl character jobvl-%data% jobvl-%offset%)
       (jobvr character jobvr-%data% jobvr-%offset%)
       (sense character sense-%data% sense-%offset%)
       (a double-float a-%data% a-%offset%)
       (wr double-float wr-%data% wr-%offset%)
       (wi double-float wi-%data% wi-%offset%)
       (vl double-float vl-%data% vl-%offset%)
       (vr double-float vr-%data% vr-%offset%)
       (scale double-float scale-%data% scale-%offset%)
       (rconde double-float rconde-%data% rconde-%offset%)
       (rcondv double-float rcondv-%data% rcondv-%offset%)
       (work double-float work-%data% work-%offset%)
       (iwork fixnum iwork-%data% iwork-%offset%)))
    (prog ((dum (make-array 1 :element-type 'double-float))
           (select (make-array 1 :element-type 't)) (anrm 0.0) (bignum 0.0)
           (cs 0.0) (cscale 0.0) (eps 0.0) (r 0.0) (scl 0.0) (smlnum 0.0)
           (sn 0.0) (hswork 0) (i 0) (icond 0) (ierr 0) (itau 0) (iwrk 0)
           (k 0) (maxb 0) (maxwrk 0) (minwrk 0) (nout 0)
           (job
            (make-array '(1) :element-type 'character :initial-element #\ ))
           (side
            (make-array '(1) :element-type 'character :initial-element #\ ))
           (lquery nil) (scalea nil) (wantvl nil) (wantvr nil) (wntsnb nil)
           (wntsne nil) (wntsnn nil) (wntsnv nil)))
      (declare (type (simple-array double-float (1)) dum)
                (type (simple-array (member t nil) (1)) select)
                (type (double-float) anrm bignum cs cscale eps r scl smlnum
                      sn)
                (type fixnum hswork i icond ierr itau iwrk k maxb
                      maxwrk minwrk nout)
                (type (simple-array character (1)) job side)
                (type (member t nil) lquery scalea wantvl wantvr wntsnb
                      wntsne wntsnn wntsnv)))

```

```

(setf info 0)
(setf lquery (coerce (= lwork -1) '(member t nil)))
(setf wantvl (char-equal jobvl #\V))
(setf wantvr (char-equal jobvr #\V))
(setf wntsnn (char-equal sense #\N))
(setf wntsne (char-equal sense #\E))
(setf wntsnv (char-equal sense #\V))
(setf wntsnb (char-equal sense #\B))
(cond
  ((not
    (or (char-equal balanc #\N)
        (char-equal balanc #\S)
        (char-equal balanc #\P)
        (char-equal balanc #\B)))
    (setf info -1))
  ((and (not wantvl) (not (char-equal jobvl #\N)))
    (setf info -2))
  ((and (not wantvr) (not (char-equal jobvr #\N)))
    (setf info -3))
  ((or (not (or wntsnn wntsne wntsnb wntsnv))
        (and (or wntsne wntsnb) (not (and wantvl wantvr))))
    (setf info -4))
  ((< n 0)
    (setf info -5))
  ((< lda (max (the fixnum 1) (the fixnum n)))
    (setf info -7))
  ((or (< ldvl 1) (and wantvl (< ldvl n)))
    (setf info -11))
  ((or (< ldvr 1) (and wantvr (< ldvr n)))
    (setf info -13))
  (setf minwrk 1)
  (cond
    ((and (= info 0) (or (>= lwork 1) lquery))
      (setf maxwrk
        (f2cl-lib:int-add n
          (f2cl-lib:int-mul n
            (ilaenv 1 "DGEHRD" " " " n
              1 n 0))))))
    (cond
      ((and (not wantvl) (not wantvr))
        (setf minwrk
          (max (the fixnum 1)
            (the fixnum (f2cl-lib:int-mul 2 n))))))
      (if (not wntsnn)
        (setf minwrk
          (max (the fixnum minwrk)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                (f2cl-lib:int-mul 6
                  n)))))))

```

[illegible]


```

n))))))
(setf maxb
  (max
    (the fixnum
      (ilaenv 8 "DHSEQR" "SN" n 1 n -1))
    (the fixnum 2)))
(setf k
  (min (the fixnum maxb)
    (the fixnum n)
    (the fixnum
      (max (the fixnum 2)
        (the fixnum
          (ilaenv 4 "DHSEQR" "EN" n 1 n -1)))))))
(setf hswrk
  (max
    (the fixnum
      (f2cl-lib:int-mul k (f2cl-lib:int-add k 2)))
    (the fixnum (f2cl-lib:int-mul 2 n))))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum 1)
    (the fixnum hswrk)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGHR" " " n 1 n
            -1)))))))
(if (and (not wntsnn) (not wntsne))
  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul n n)
          (f2cl-lib:int-mul 6
            n))))))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum (f2cl-lib:int-mul 3 n))
    (the fixnum 1))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (the fixnum maxwrk) 'double-float)))
(cond
  ((and (< lwork minwrk) (not lquery))
    (setf info -21)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~a"

```

```

        "DGEVX" (f2cl-lib:int-sub info))
      (go end_label))
    (lquery
      (go end_label)))
    (if (= n 0) (go end_label))
    (setf eps (dlamch "P"))
    (setf smlnum (dlamch "S"))
    (setf bignum (/ one smlnum))
    (multiple-value-bind (var-0 var-1)
      (dlabad smlnum bignum)
      (declare (ignore))
      (setf smlnum var-0)
      (setf bignum var-1))
    (setf smlnum (/ (f2cl-lib:fsqrt smlnum) eps))
    (setf bignum (/ one smlnum))
    (setf icond 0)
    (setf anrm (dlange "M" n n a lda dum))
    (setf scalea nil)
    (cond
      ((and (> anrm zero) (< anrm smlnum))
        (setf scalea t)
        (setf cscale smlnum))
      ((> anrm bignum)
        (setf scalea t)
        (setf cscale bignum)))
    (if scalea
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
        (dlascl "G" 0 0 anrm cscale n n a lda ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8))
        (setf ierr var-9)))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgebal balanc n a lda ilo ihi scale ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-6))
        (setf ilo var-4)
        (setf ihi var-5)
        (setf ierr var-7))
        (setf abnrm (dlange "1" n n a lda dum))
        (cond
          (scalea
            (setf (f2cl-lib:fref dum (1) ((1 1))) abnrm)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
              (dlascl "G" 0 0 cscale anrm 1 1 dum 1 ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                var-8))
              (setf ierr var-9))
              (setf abnrm (f2cl-lib:fref dum (1) ((1 1))))))
          (setf itau 1)

```

```

(setf iwrk (f2cl-lib:int-add itau n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (dgehrd n ilo ihi a lda
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
  (setf ierr var-8))
(cond
  (wantvl
    (f2cl-lib:f2cl-set-string side "L" (string 1))
    (dlacpy "L" n n a lda vl ldvl)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (dorghr n ilo ihi vl ldvl
        (f2cl-lib:array-slice work double-float (itau) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
      (setf ierr var-8))
    (setf iwrk itau)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13)
      (dhseqr "S" "V" n ilo ihi a lda wr wi vl ldvl
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12))
      (setf info var-13))
    (cond
      (wantvr
        (f2cl-lib:f2cl-set-string side "B" (string 1))
        (dlacpy "F" n n vl ldvl vr ldvr))))
  (wantvr
    (f2cl-lib:f2cl-set-string side "R" (string 1))
    (dlacpy "L" n n a lda vr ldvr)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (dorghr n ilo ihi vr ldvr
        (f2cl-lib:array-slice work double-float (itau) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
      (setf ierr var-8))
    (setf iwrk itau)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13)

```

```

        (dhseqr "S" "V" n ilo ihi a lda wr wi vr ldvr
          (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10 var-11 var-12))
        (setf info var-13)))
      (t
        (cond
          (wntsn
            (f2cl-lib:f2cl-set-string job "E" (string 1)))
          (t
            (f2cl-lib:f2cl-set-string job "S" (string 1))))
        (setf iwrk itau)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
            var-10 var-11 var-12 var-13)
          (dhseqr job "N" n ilo ihi a lda wr wi vr ldvr
            (f2cl-lib:array-slice work double-float (iwrk) ((1 *)))
            (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
              var-8 var-9 var-10 var-11 var-12))
            (setf info var-13))))
        (if (> info 0) (go label50))
        (cond
          ((or wantvl wantvr)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11 var-12 var-13)
              (dtrevc side "B" select n a lda vl ldvl vr ldvr n nout
                (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) ierr)
                (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10 var-12))
                (setf nout var-11)
                (setf ierr var-13))))
          (cond
            (not wntsn)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17)
              (dtrsna sense "A" select n a lda vl ldvl vr ldvr rconde rcondv n
                nout (f2cl-lib:array-slice work double-float (iwrk) ((1 *))) n
                  iwork icond)
                (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10 var-11 var-12 var-14 var-15
                    var-16))
                (setf nout var-13)
                (setf icond var-17))))
            (cond
              (wantvl
                (multiple-value-bind

```

```

(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
(dgebak balanc "L" n ilo ihi scale n vl ldvl ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8))
(setf ierr var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
((> i n) nil)
(tagbody
(cond
((= (f2cl-lib:fref wi (i) ((1 *))) zero)
(setf scl
(/ one
(dnrm2 n
(f2cl-lib:array-slice vl
double-float
(1 i)
((1 ldvl) (1 *)))
1)))
(dscal n scl
(f2cl-lib:array-slice vl
double-float
(1 i)
((1 ldvl) (1 *)))
1))
(> (f2cl-lib:fref wi (i) ((1 *))) zero)
(setf scl
(/ one
(dlapy2
(dnrm2 n
(f2cl-lib:array-slice vl
double-float
(1 i)
((1 ldvl) (1 *)))
1)
(dnrm2 n
(f2cl-lib:array-slice vl
double-float
(1 (f2cl-lib:int-add i 1))
((1 ldvl) (1 *)))
1))))
(dscal n scl
(f2cl-lib:array-slice vl
double-float
(1 i)
((1 ldvl) (1 *)))
1)
(dscal n scl
(f2cl-lib:array-slice vl
double-float
(1 (f2cl-lib:int-add i 1))

```

```

((1 ldvl) (1 *)))
1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
    (k)
    ((1 *))
    work-%offset%)
    (+
      (expt
        (f2cl-lib:fref vl-%data%
          (k i)
          ((1 ldvl) (1 *))
          vl-%offset%)
        2)
      (expt
        (f2cl-lib:fref vl-%data%
          (k (f2cl-lib:int-add i 1))
          ((1 ldvl) (1 *))
          vl-%offset%)
        2))))))
(setf k (idamax n work 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg
    (f2cl-lib:fref vl-%data%
      (k i)
      ((1 ldvl) (1 *))
      vl-%offset%)
    (f2cl-lib:fref vl-%data%
      (k (f2cl-lib:int-add i 1))
      ((1 ldvl) (1 *))
      vl-%offset%)
    cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(drot n
  (f2cl-lib:array-slice vl
    double-float
    (1 i)
    ((1 ldvl) (1 *)))
  1
  (f2cl-lib:array-slice vl
    double-float
    (1 (f2cl-lib:int-add i 1))
    ((1 ldvl) (1 *)))
  1 cs sn)
(setf (f2cl-lib:fref vl-%data%

```

```

                                (k (f2cl-lib:int-add i 1))
                                ((1 ldvl) (1 *))
                                vl-%offset%)
                                zero))))))
(cond
 (wantvr
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
   (dgebak balanc "R" n ilo ihi scale n vr ldvr ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8))
   (setf ierr var-9))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
   (cond
    ((= (f2cl-lib:fref wi (i) ((1 *))) zero)
     (setf scl
              (/ one
                 (dnrm2 n
                      (f2cl-lib:array-slice vr
                                               double-float
                                               (1 i)
                                               ((1 ldvr) (1 *)))
                      1)))
     (dscal n scl
            (f2cl-lib:array-slice vr
                                   double-float
                                   (1 i)
                                   ((1 ldvr) (1 *)))
            1))
    (> (f2cl-lib:fref wi (i) ((1 *))) zero)
     (setf scl
              (/ one
                 (dlapy2
                  (dnrm2 n
                       (f2cl-lib:array-slice vr
                                                double-float
                                                (1 i)
                                                ((1 ldvr) (1 *)))
                       1)
                  (dnrm2 n
                       (f2cl-lib:array-slice vr
                                                double-float
                                                (1 (f2cl-lib:int-add i 1))
                                                ((1 ldvr) (1 *)))
                       1))))
     (dscal n scl
            (f2cl-lib:array-slice vr
                                   double-float

```

```

                                (1 i)
                                ((1 ldvr) (1 *)))
1)
(dscal n scl
  (f2cl-lib:array-slice vr
    double-float
    (1 (f2cl-lib:int-add i 1))
    ((1 ldvr) (1 *)))
1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                        (k)
                        ((1 *)
                        work-%offset%)
    (+
      (expt
        (f2cl-lib:fref vr-%data%
                        (k i)
                        ((1 ldvr) (1 *)
                        vr-%offset%)
      2)
      (expt
        (f2cl-lib:fref vr-%data%
                        (k (f2cl-lib:int-add i 1))
                        ((1 ldvr) (1 *)
                        vr-%offset%)
      2))))))
(setf k (idamax n work 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg
    (f2cl-lib:fref vr-%data%
                    (k i)
                    ((1 ldvr) (1 *)
                    vr-%offset%)
    (f2cl-lib:fref vr-%data%
                    (k (f2cl-lib:int-add i 1))
                    ((1 ldvr) (1 *)
                    vr-%offset%)
    cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(drot n
  (f2cl-lib:array-slice vr
    double-float
    (1 i)
    ((1 ldvr) (1 *)))

```



```

1
(f2cl-lib:array-slice vr
                        double-float
                        (1 (f2cl-lib:int-add i 1))
                        ((1 ldvr) (1 *)))

1 cs sn)
(setf (f2cl-lib:fref vr-%data%
                    (k (f2cl-lib:int-add i 1))
                    ((1 ldvr) (1 *))
                    vr-%offset%)
      zero))))))

label50
(cond
 (scalea
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
   (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
    (f2cl-lib:array-slice wr double-float ((+ info 1)) ((1 *)))
    (max (the fixnum (f2cl-lib:int-sub n info))
         (the fixnum 1))
    ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8))
   (setf ierr var-9))
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
   (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
    (f2cl-lib:array-slice wi double-float ((+ info 1)) ((1 *)))
    (max (the fixnum (f2cl-lib:int-sub n info))
         (the fixnum 1))
    ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8))
   (setf ierr var-9))
  (cond
   ((= info 0)
    (if (and (or wntsnv wntsnb) (= icond 0))
        (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
         (dlascl "G" 0 0 cscale anrm n 1 rcondv n ierr)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                          var-7 var-8))
         (setf ierr var-9))))
    (t
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wr n
       ierr)

```

```

      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8))
      (setf ierr var-9))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 wi n
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8))
      (setf ierr var-9))))))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum maxwrk) 'double-float))
  end_label
  (return
    (values nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            nil
            ilo
            ihi
            nil
            abnrm
            nil
            nil
            nil
            nil
            nil
            info))))))

```

dgehd2 LAPACK

— dgehd2.input —

```
)set break resume
```

```

)sys rm -f dgehd2.output
)spool dgehd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgehd2.help —

```

=====
dgehd2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DGEHD2 - a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation

SYNOPSIS

```
SUBROUTINE DGEHD2( N, ILO, IHI, A, LDA, TAU, WORK, INFO )
```

```
      INTEGER      IHI, ILO, INFO, LDA, N
```

```
      DOUBLE      PRECISION A( LDA, * ), TAU( * ), WORK( * )
```

PURPOSE

DGEHD2 reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q' * A * Q = H$.

ARGUMENTS

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

ILO (input) INTEGER
IHI (input) INTEGER It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the n by n general matrix to be reduced. On exit,

the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1, N)$.

TAU (output) DOUBLE PRECISION array, dimension (N-1)
The scalar factors of the elementary reflectors (see Further Details).

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = -i, the i-th argument had an illegal value.

FURTHER DETAILS

The matrix Q is represented as a product of (ihi-ilo) elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with $v(1:i) = 0$, $v(i+1) = 1$ and $v(ihi+1:n) = 0$; $v(i+2:ihi)$ is stored on exit in $A(i+2:ihi, i)$, and tau in $TAU(i)$.

The contents of A are illustrated by the following example, with $n = 7$, $ilo = 2$ and $ihi = 6$:

| on entry, | on exit, |
|-------------------|--------------------|
| (a a a a a a a) | (a a h h h h a) |
| (a a a a a a a) | (a h h h h h a) |
| (a a a a a a a) | (h h h h h h) |
| (a a a a a a a) | (v2 h h h h h) |
| (a a a a a a a) | (v2 v3 h h h h) |
| (a a a a a a a) | (v2 v3 v4 h h h) |
| (a a a a a a a) | (a a a a a a a) |

where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and v_i denotes an element of the vector defining H(i).

— dgehd2.f —

```

SUBROUTINE DGEHD2( N, ILO, IHI, A, LDA, TAU, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1992
*
*  .. Scalar Arguments ..
INTEGER          IHI, ILO, INFO, LDA, N
*
*  ..
*  .. Array Arguments ..
DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION ONE
PARAMETER        ( ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
INTEGER          I
DOUBLE PRECISION AII
*
*  ..
*  .. External Subroutines ..
EXTERNAL          DLARF, DLARFG, XERBLA
*
*  ..
*  .. Intrinsic Functions ..
INTRINSIC         MAX, MIN
*
*  ..
*  .. Executable Statements ..
*
*  Test the input parameters
*
*
INFO = 0
IF( N.LT.0 ) THEN
    INFO = -1
ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
    INFO = -2
ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
    INFO = -3
ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
    INFO = -5
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DGEHD2', -INFO )
    RETURN

```

```

      END IF
*
      DO 10 I = ILO, IHI - 1
*
*       Compute elementary reflector H(i) to annihilate A(i+2:ihi,i)
*
      CALL DLARFG( IHI-I, A( I+1, I ), A( MIN( I+2, N ), I ), 1,
$              TAU( I ) )
      AII = A( I+1, I )
      A( I+1, I ) = ONE
*
*       Apply H(i) to A(1:ihi,i+1:ihi) from the right
*
      CALL DLARF( 'Right', IHI, IHI-I, A( I+1, I ), 1, TAU( I ),
$              A( 1, I+1 ), LDA, WORK )
*
*       Apply H(i) to A(i+1:ihi,i+1:n) from the left
*
      CALL DLARF( 'Left', IHI-I, N-I, A( I+1, I ), 1, TAU( I ),
$              A( I+1, I+1 ), LDA, WORK )
*
      A( I+1, I ) = AII
10 CONTINUE
*
      RETURN
*
*       End of DGEHD2
*
      END

```

— LAPACK dgehd2 —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgehd2 (n ilo ihi a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
      (type fixnum info lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0))
        (declare (type (double-float) aii) (type fixnum i))
        (setf info 0)
        (cond
          ((< n 0)

```

```

      (setf info -1))
    ((or (< ilo 1)
         (> ilo
            (max (the fixnum 1) (the fixnum n))))
      (setf info -2))
    ((or
      (< ihi (min (the fixnum ilo) (the fixnum n)))
      (> ihi n))
      (setf info -3))
    ((< lda (max (the fixnum 1) (the fixnum n)))
      (setf info -5)))
  (cond
    ((/= info 0)
     (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGEHD2" (f2cl-lib:int-sub info))
     (go end_label)))
  (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
    ((> i (f2cl-lib:int-add ihi (f2cl-lib:int-sub 1)))) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlarf (f2cl-lib:int-sub ihi i)
        (f2cl-lib:fref a-%data%
          ((f2cl-lib:int-add i 1) i)
          ((1 lda) (1 *))
          a-%offset%)
        (f2cl-lib:array-slice a
          double-float
          ((min (f2cl-lib:int-add i 2) n) i)
          ((1 lda) (1 *)))
        1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
      (declare (ignore var-0 var-2 var-3))
      (setf (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add i 1) i)
        ((1 lda) (1 *))
        a-%offset%)
        var-1)
      (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
    (setf aii
      (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add i 1) i)
        ((1 lda) (1 *))
        a-%offset%))
    (setf (f2cl-lib:fref a-%data%
      ((f2cl-lib:int-add i 1) i)
      ((1 lda) (1 *))
      a-%offset%)
      one)
    (dlarf "Right" ihi (f2cl-lib:int-sub ihi i)
      (f2cl-lib:array-slice a double-float ((+ i 1) i) ((1 lda) (1 *)))

```

```

1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
(f2cl-lib:array-slice a
  double-float
  (1 (f2cl-lib:int-add i 1))
  ((1 lda) (1 *)))
lda work)
(dlarf "Left" (f2cl-lib:int-sub ihi i) (f2cl-lib:int-sub n i)
(f2cl-lib:array-slice a double-float ((+ i 1) i) ((1 lda) (1 *)))
1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
(f2cl-lib:array-slice a
  double-float
  ((+ i 1) (f2cl-lib:int-add i 1))
  ((1 lda) (1 *)))
lda work)
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-add i 1) i)
  ((1 lda) (1 *))
  a-%offset%)
  aii)))
end_label
(return (values nil nil nil nil nil nil nil info))))

```

dgehrd LAPACK

— dgehrd.input —

```

)set break resume
)sys rm -f dgehrd.output
)spool dgehrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgehrd.help —

```

=====
dgehrd examples
=====

```



```
=====
Man Page Details
=====
```

NAME

DGEHRD - Reduce a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation

SYNOPSIS

```
SUBROUTINE DGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
```

```
      INTEGER          IHI, ILO, INFO, LDA, LWORK, N
      DOUBLE            PRECISION A( LDA, * ), TAU( * ), WORK( * )
```

PURPOSE

DGEHRD reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation: $Q' * A * Q = H$.

ARGUMENTS

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

ILO (input) INTEGER
IHI (input) INTEGER It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the N-by-N general matrix to be reduced. On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,N)$.

TAU (output) DOUBLE PRECISION array, dimension (N-1)
The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK. $LWORK \geq \max(1,N)$. For optimum performance $LWORK \geq N * NB$, where NB is the optimal blocksize.

If `LWORK = -1`, then a workspace query is assumed; the routine only calculates the optimal size of the `WORK` array, returns this value as the first entry of the `WORK` array, and no error message related to `LWORK` is issued by `XERBLA`.

`INFO` (output) INTEGER
 = 0: successful exit
 < 0: if `INFO = -i`, the *i*-th argument had an illegal value.

FURTHER DETAILS

The matrix `Q` is represented as a product of (*ihi-ilo*) elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each `H(i)` has the form

$$H(i) = I - \tau * v * v'$$

where `tau` is a real scalar, and `v` is a real vector with `v(1:i) = 0`, `v(i+1) = 1` and `v(ihi+1:n) = 0`; `v(i+2:ihi)` is stored on exit in `A(i+2:ihi,i)`, and `tau` in `TAU(i)`.

The contents of `A` are illustrated by the following example, with `n = 7`, `ilo = 2` and `ihi = 6`:

| on entry, | on exit, |
|-------------------|--------------------|
| (a a a a a a a) | (a a h h h h a) |
| (a a a a a a a) | (a h h h h a) |
| (a a a a a a a) | (h h h h h h) |
| (a a a a a a a) | (v2 h h h h h) |
| (a a a a a a a) | (v2 v3 h h h h) |
| (a a a a a a a) | (v2 v3 v4 h h h) |
| (a a a a a a a) | (a a a a a a) |

where `a` denotes an element of the original matrix `A`, `h` denotes a modified element of the upper Hessenberg matrix `H`, and `vi` denotes an element of the vector defining `H(i)`.

See Quintana-Orti and Van de Geijn (2005).

— dgehrd.f —

SUBROUTINE DGEHRD(N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO)

```

*
* -- LAPACK routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   June 30, 1999
*
*   .. Scalar Arguments ..
*   INTEGER          IHI, ILO, INFO, LDA, LWORK, N
*   ..
*   .. Array Arguments ..
*   DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*   ..
*
* =====
*
*   .. Parameters ..
*   INTEGER          NBMAX, LDT
*   PARAMETER        ( NBMAX = 64, LDT = NBMAX+1 )
*   DOUBLE PRECISION ZERO, ONE
*   PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*   ..
*   .. Local Scalars ..
*   LOGICAL          LQUERY
*   INTEGER          I, IB, IINFO, IWS, LDWORK, LWKOPT, NB, NBMIN,
*   $                NH, NX
*   DOUBLE PRECISION EI
*   ..
*   .. Local Arrays ..
*   DOUBLE PRECISION T( LDT, NBMAX )
*   ..
*   .. External Subroutines ..
*   EXTERNAL         DGEHD2, DGEMM, DLAHRD, DLARFB, XERBLA
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC        MAX, MIN
*   ..
*   .. External Functions ..
*   INTEGER          ILAENV
*   EXTERNAL         ILAENV
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters
*
*   INFO = 0
*   NB = MIN( NBMAX, ILAENV( 1, 'DGEHRD', ' ', N, ILO, IHI, -1 ) )
*   LWKOPT = N*NB
*   WORK( 1 ) = LWKOPT
*   LQUERY = ( LWORK.EQ.-1 )
*   IF( N.LT.0 ) THEN

```

```

        INFO = -1
    ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
        INFO = -2
    ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -3
    ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -5
    ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
        INFO = -8
    END IF
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DGEHRD', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Set elements 1:ILO-1 and IHI:N-1 of TAU to zero
*
    DO 10 I = 1, ILO - 1
        TAU( I ) = ZERO
10  CONTINUE
    DO 20 I = MAX( 1, IHI ), N - 1
        TAU( I ) = ZERO
20  CONTINUE
*
*   Quick return if possible
*
    NH = IHI - ILO + 1
    IF( NH.LE.1 ) THEN
        WORK( 1 ) = 1
        RETURN
    END IF
*
*   Determine the block size.
*
    NB = MIN( NBMAX, ILAENV( 1, 'DGEHRD', ' ', N, ILO, IHI, -1 ) )
    NBMIN = 2
    IWS = 1
    IF( NB.GT.1 .AND. NB.LT.NH ) THEN
*
*       Determine when to cross over from blocked to unblocked code
*       (last block is always handled by unblocked code).
*
        NX = MAX( NB, ILAENV( 3, 'DGEHRD', ' ', N, ILO, IHI, -1 ) )
        IF( NX.LT.NH ) THEN
*
*           Determine if workspace is large enough for blocked code.
*
            IWS = N*NB

```

```

      IF( LWORK.LT.IWS ) THEN
*
*       Not enough workspace to use optimal NB:  determine the
*       minimum value of NB, and reduce NB or force use of
*       unblocked code.
*
      NBMIN = MAX( 2, ILAENV( 2, 'DGEHRD', ' ', N, ILO, IHI,
$          -1 ) )
      IF( LWORK.GE.N*N*NBMIN ) THEN
          NB = LWORK / N
      ELSE
          NB = 1
      END IF
      END IF
      LDWORK = N
*
      IF( NB.LT.NBMIN .OR. NB.GE.NH ) THEN
*
*       Use unblocked code below
*
          I = ILO
*
      ELSE
*
*       Use blocked code
*
          DO 30 I = ILO, IHI - 1 - NX, NB
              IB = MIN( NB, IHI-I )
*
*       Reduce columns i:i+ib-1 to Hessenberg form, returning the
*       matrices V and T of the block reflector  $H = I - V*T*V'$ 
*       which performs the reduction, and also the matrix  $Y = A*V*T$ 
*
          CALL DLAHRD( IHI, I, IB, A( 1, I ), LDA, TAU( I ), T, LDT,
$              WORK, LDWORK )
*
*       Apply the block reflector H to A(1:ihi,i+ib:ihi) from the
*       right, computing  $A := A - Y * V'$ . V(i+ib,ib-1) must be set
*       to 1.
*
          EI = A( I+IB, I+IB-1 )
          A( I+IB, I+IB-1 ) = ONE
          CALL DGEMM( 'No transpose', 'Transpose', IHI, IHI-I-IB+1,
$              IB, -ONE, WORK, LDWORK, A( I+IB, I ), LDA, ONE,
$              A( 1, I+IB ), LDA )
          A( I+IB, I+IB-1 ) = EI
*
*       Apply the block reflector H to A(i+1:ihi,i+ib:n) from the

```

```

*           left
*
*           CALL DLARFB( 'Left', 'Transpose', 'Forward', 'Columnwise',
$             IHI-I, N-I-IB+1, IB, A( I+1, I ), LDA, T, LDT,
$             A( I+1, I+IB ), LDA, WORK, LDWORK )
30    CONTINUE
      END IF
*
*    Use unblocked code to reduce the rest of the matrix
*
*    CALL DGEHD2( N, I, IHI, A, LDA, TAU, WORK, IINFO )
*    WORK( 1 ) = IWS
*
*    RETURN
*
*    End of DGEHRD
*
*    END

```

— LAPACK dgehrd —

```

(let* ((nbmax 64) (ldt (+ nbmax 1)) (zero 0.0) (one 1.0))
  (declare (type (fixnum 64 64) nbmax)
            (type fixnum ldt)
            (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgehrd (n ilo ihi a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lwork lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((ei 0.0) (i 0) (ib 0) (iinfo 0) (iws 0) (ldwork 0) (lwkopt 0)
              (nb 0) (nbmin 0) (nh 0) (nx 0) (lquery nil)
              (t$
                (make-array (the fixnum (reduce #'* (list ldt nbmax)))
                             :element-type 'double-float)))
              (declare (type (simple-array double-float (*)) t$)
                        (type (double-float) ei)
                        (type fixnum i ib iinfo iws ldwork lwkopt nb
                                nbmin nh nx)
                        (type (member t nil) lquery))
              (setf info 0)
              (setf nb
                (min (the fixnum nbmax)

```

```

        (the fixnum
          (ilaenv 1 "DGEHRD" " " n ilo ihi -1))))
(setf lwkopt (f2cl-lib:int-mul n nb))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum lwkopt) 'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((< n 0)
   (setf info -1))
  ((or (< ilo 1)
       (> ilo
        (max (the fixnum 1) (the fixnum n))))
   (setf info -2))
  ((or
    (< ihi (min (the fixnum ilo) (the fixnum n)))
    (> ihi n))
   (setf info -3))
  ((< lda (max (the fixnum 1) (the fixnum n)))
   (setf info -5))
  ((and
    (< lwork (max (the fixnum 1) (the fixnum n)))
    (not lquery))
   (setf info -8)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGEHRD" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add ilo (f2cl-lib:int-sub 1))) nil)
  (tagbody
   (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) zero)))
(f2cl-lib:fdo (i
  (max (the fixnum 1)
        (the fixnum ihi))
  (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
   (setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) zero)))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(cond
  ((<= nh 1)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nb
  (min (the fixnum nbmax)

```

```

        (the fixnum
          (ilaenv 1 "DGEHRD" " " n ilo ihi -1)))
(setf nbmin 2)
(setf iws 1)
(cond
  ((and (> nb 1) (< nb nh))
    (setf nx
      (max (the fixnum nb)
        (the fixnum
          (ilaenv 3 "DGEHRD" " " n ilo ihi -1))))
    (cond
      ((< nx nh)
        (setf iws (f2cl-lib:int-mul n nb))
        (cond
          ((< lwork iws)
            (setf nbmin
              (max (the fixnum 2)
                (the fixnum
                  (ilaenv 2 "DGEHRD" " " n ilo ihi -1))))
            (cond
              ((>= lwork (f2cl-lib:int-mul n nbmin))
                (setf nb (the fixnum (truncate lwork n))))
              (t
                (setf nb 1))))))
          (setf nb 1))))))
(setf ldwork n)
(cond
  ((or (< nb nbmin) (>= nb nh))
    (setf i ilo))
  (t
    (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i nb))
      ((> i
        (f2cl-lib:int-add ihi
          (f2cl-lib:int-sub 1)
          (f2cl-lib:int-sub nx)))
        nil)
      (tagbody
        (setf ib
          (min (the fixnum nb)
            (the fixnum (f2cl-lib:int-sub ihi i))))
        (dlahrd ihi i ib
          (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) lda
          (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt work
          ldwork)
        (setf ei
          (f2cl-lib:fref a-%data%
            ((f2cl-lib:int-add i ib)
              (f2cl-lib:int-sub
                (f2cl-lib:int-add i ib)
                1))
              ((1 lda) (1 *)))

```



```

                                a-%offset%))
(setf (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add i ib)
                     (f2cl-lib:int-sub (f2cl-lib:int-add i ib)
                                         1))
                    ((1 lda) (1 *))
                    a-%offset%))

    one)
(dgemm "No transpose" "Transpose" ihi
 (f2cl-lib:int-add (f2cl-lib:int-sub ihi i ib) 1) ib (- one)
 work ldwork
 (f2cl-lib:array-slice a
  double-float
  ((+ i ib) i)
  ((1 lda) (1 *))))

lda one
(f2cl-lib:array-slice a
 double-float
 (1 (f2cl-lib:int-add i ib))
 ((1 lda) (1 *)))

lda)
(setf (f2cl-lib:fref a-%data%
                    ((f2cl-lib:int-add i ib)
                     (f2cl-lib:int-sub (f2cl-lib:int-add i ib)
                                         1))
                    ((1 lda) (1 *))
                    a-%offset%))

    ei)
(dlarfb "Left" "Transpose" "Forward" "Columnwise"
 (f2cl-lib:int-sub ihi i)
 (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
 (f2cl-lib:array-slice a
  double-float
  ((+ i 1) i)
  ((1 lda) (1 *))))

lda t$ ldt
(f2cl-lib:array-slice a
 double-float
 ((+ i 1) (f2cl-lib:int-add i ib))
 ((1 lda) (1 *)))

lda work ldwork))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgehd2 n i ihi a lda tau work iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf iinfo var-7))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum iws) 'double-float))

end_label
(return (values nil nil nil nil nil nil nil nil info))))))

```

dgelq2 LAPACK

— dgelq2.input —

```

)set break resume
)sys rm -f dgelq2.output
)spool dgelq2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgelq2.help —

```
=====
dgelq2 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGELQ2 - an LQ factorization of a real m by n matrix A

SYNOPSIS

```
SUBROUTINE DGELQ2( M, N, A, LDA, TAU, WORK, INFO )
```

```
      INTEGER          INFO, LDA, M, N
```

```
      DOUBLE           PRECISION A( LDA, * ), TAU( * ), WORK( * )
```

PURPOSE

DGELQ2 computes an LQ factorization of a real m by n matrix A: $A = L * Q$.

ARGUMENTS

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the m by n matrix A. On exit, the elements on and below the diagonal of the array contain the m by $\min(m,n)$ lower trapezoidal matrix L (L is lower triangular if $m \leq n$); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details). LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,M)$.

TAU (output) DOUBLE PRECISION array, dimension (min(M,N))
The scalar factors of the elementary reflectors (see Further Details).

WORK (workspace) DOUBLE PRECISION array, dimension (M)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

FURTHER DETAILS

The matrix Q is represented as a product of elementary reflectors

$$Q = H(k) \dots H(2) H(1), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with
 $v(1:i-1) = 0$ and $v(i) = 1$; $v(i+1:n)$ is stored on exit in $A(i,i+1:n)$,
 and tau in TAU(i).

— dgelq2.f —

SUBROUTINE DGELQ2(M, N, A, LDA, TAU, WORK, INFO)

```
*
* -- LAPACK routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University
* February 29, 1992
*
* .. Scalar Arguments ..
```

```

      INTEGER          INFO, LDA, M, N
*
*   ..
*   .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), TAU( * ), WORK( * )
*
*   ..
*
*   =====
*
*   .. Parameters ..
      DOUBLE PRECISION  ONE
      PARAMETER          ( ONE = 1.0D+0 )
*
*   ..
*   .. Local Scalars ..
      INTEGER           I, K
      DOUBLE PRECISION  AII
*
*   ..
*   .. External Subroutines ..
      EXTERNAL          DLARF, DLARFG, XERBLA
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGELQ2', -INFO )
         RETURN
      END IF
*
      K = MIN( M, N )
*
      DO 10 I = 1, K
*
*         Generate elementary reflector H(i) to annihilate A(i,i+1:n)
*
*         CALL DLARFG( N-I+1, A( I, I ), A( I, MIN( I+1, N ) ), LDA,
$           TAU( I ) )
*         IF( I.LT.M ) THEN
*
*            Apply H(i) to A(i+1:m,i:n) from the right

```

```

*
      AII = A( I, I )
      A( I, I ) = ONE
      CALL DLARF( 'Right', M-I, N-I+1, A( I, I ), LDA, TAU( I ),
$          A( I+1, I ), LDA, WORK )
      A( I, I ) = AII
    END IF
10 CONTINUE
    RETURN
*
*    End of DGELQ2
*
    END

```

— LAPACK dgelq2 —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgelq2 (m n a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (k 0))
        (declare (type (double-float) aii) (type fixnum i k))
        (setf info 0)
        (cond
          ((< m 0)
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -4)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DGELQ2" (f2cl-lib:int-sub info))
           (go end_label)))
        (setf k (min (the fixnum m) (the fixnum n)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i k) nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)

```

```

(dlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
  (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
  (f2cl-lib:array-slice a
    double-float
    (i
      (min
        (the fixnum
          (f2cl-lib:int-add i 1))
        (the fixnum n)))
    ((1 lda) (1 *)))
  lda (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
  var-1)
(setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
(cond
  (< i m)
  (setf aii
    (f2cl-lib:fref a-%data%
      (i i)
      ((1 lda) (1 *))
      a-%offset%))
    (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
      one)
    (dlarf "Right" (f2cl-lib:int-sub m i)
      (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
      (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
      (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
      (f2cl-lib:array-slice a
        double-float
        ((+ i 1) i)
        ((1 lda) (1 *)))
      lda work)
    (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
      aii))))
end_label
(return (values nil nil nil nil nil nil info))))

```

dgelqf LAPACK

— dgelqf.input —

```

)set break resume
)sys rm -f dgelqf.output

```

```
)spool dgelqf.output
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dgelqf.help —

```
=====
dgelqf examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGELQF - an LQ factorization of a real M-by-N matrix A

SYNOPSIS

SUBROUTINE DGELQF(M, N, A, LDA, TAU, WORK, LWORK, INFO)

INTEGER INFO, LDA, LWORK, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

PURPOSE

DGELQF computes an LQ factorization of a real M-by-N matrix A: $A = L * Q$.

ARGUMENTS

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if $m \leq n$); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details). LDA (input) INTEGER The leading dimen-

sion of the array A. $LDA \geq \max(1, M)$.

TAU (output) DOUBLE PRECISION array, dimension $(\min(M, N))$
The scalar factors of the elementary reflectors (see Further Details).

WORK (workspace/output) DOUBLE PRECISION array, dimension $(\max(1, LWORK))$

On exit, if $INFO = 0$, $WORK(1)$ returns the optimal $LWORK$.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1, M)$. For optimum performance $LWORK \geq M \cdot NB$, where NB is the optimal block-size.

If $LWORK = -1$, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if $INFO = -i$, the i -th argument had an illegal value

FURTHER DETAILS

The matrix Q is represented as a product of elementary reflectors

$$Q = H(k) \dots H(2) H(1), \text{ where } k = \min(m, n).$$

Each $H(i)$ has the form

$$H(i) = I - \tau * v * v'$$

where τ is a real scalar, and v is a real vector with $v(1:i-1) = 0$ and $v(i) = 1$; $v(i+1:n)$ is stored on exit in $A(i, i+1:n)$, and τ in $TAU(i)$.

—————

— dgelqf.f —

SUBROUTINE DGELQF(M, N, A, LDA, TAU, WORK, LWORK, INFO)

```
*
* -- LAPACK routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University
* June 30, 1999
*
```



```

*      .. Scalar Arguments ..
      INTEGER          INFO, LDA, LWORK, M, N
*
*      ..
*      .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*
*      ..
*
*      =====
*
*      .. Local Scalars ..
      LOGICAL          LQUERY
      INTEGER          I, IB, IINFO, IWS, K, LDWORK, LWKOPT, NB,
      $                NBMIN, NX
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DGELQ2, DLARFB, DLARFT, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*
*      ..
*      .. External Functions ..
      INTEGER          ILAENV
      EXTERNAL          ILAENV
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      NB = ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      LWKOPT = M*NB
      WORK( 1 ) = LWKOPT
      LQUERY = ( LWORK.EQ.-1 )
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      ELSE IF( LWORK.LT.MAX( 1, M ) .AND. .NOT.LQUERY ) THEN
         INFO = -7
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGELQF', -INFO )
         RETURN
      ELSE IF( LQUERY ) THEN
         RETURN
      END IF
*
*      Quick return if possible

```

```

*
      K = MIN( M, N )
      IF( K.EQ.0 ) THEN
          WORK( 1 ) = 1
          RETURN
      END IF
*
      NBMIN = 2
      NX = 0
      IWS = M
      IF( NB.GT.1 .AND. NB.LT.K ) THEN
*
*         Determine when to cross over from blocked to unblocked code.
*
*         NX = MAX( 0, ILAENV( 3, 'DGELQF', ' ', M, N, -1, -1 ) )
*         IF( NX.LT.K ) THEN
*
*             Determine if workspace is large enough for blocked code.
*
*             LDWORK = M
*             IWS = LDWORK*NB
*             IF( LWORK.LT.IWS ) THEN
*
*                 Not enough workspace to use optimal NB:  reduce NB and
*                 determine the minimum value of NB.
*
*                 NB = LWORK / LDWORK
*                 NBMIN = MAX( 2, ILAENV( 2, 'DGELQF', ' ', M, N, -1,
$                   -1 ) )
*             END IF
*         END IF
*     END IF
*
*     IF( NB.GE.NBMIN .AND. NB.LT.K .AND. NX.LT.K ) THEN
*
*         Use blocked code initially
*
*         DO 10 I = 1, K - NX, NB
*             IB = MIN( K-I+1, NB )
*
*             Compute the LQ factorization of the current block
*             A(i:i+ib-1,i:n)
*
*             CALL DGELQ2( IB, N-I+1, A( I, I ), LDA, TAU( I ), WORK,
$               IINFO )
*             IF( I+IB.LE.M ) THEN
*
*                 Form the triangular factor of the block reflector
*                 H = H(i) H(i+1) . . . H(i+ib-1)
*

```

```

                CALL DLARFT( 'Forward', 'Rowwise', N-I+1, IB, A( I, I ),
$                LDA, TAU( I ), WORK, LDWORK )
*
*                Apply H to A(i+ib:m,i:n) from the right
*
                CALL DLARFB( 'Right', 'No transpose', 'Forward',
$                'Rowwise', M-I-IB+1, N-I+1, IB, A( I, I ),
$                LDA, WORK, LDWORK, A( I+IB, I ), LDA,
$                WORK( IB+1 ), LDWORK )
                END IF
10    CONTINUE
    ELSE
        I = 1
    END IF
*
*    Use unblocked code to factor the last or only block.
*
    IF( I.LE.K )
$    CALL DGELQ2( M-I+1, N-I+1, A( I, I ), LDA, TAU( I ), WORK,
$                IINFO )
*
    WORK( 1 ) = IWS
    RETURN
*
*    End of DGELQF
*
END

```

— LAPACK dgelqf —

```

(defun dgelqf (m n a lda tau work lwork info)
  (declare (type (simple-array double-float (*)) work tau a)
            (type fixnum info lwork lda n m))
  (f2cl-lib:with-multi-array-data
    ((a double-float a-%data% a-%offset%)
     (tau double-float tau-%data% tau-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (k 0) (ldwork 0) (lwkopt 0) (nb 0)
           (nbmin 0) (nx 0) (lquery nil))
      (declare (type (member t nil) lquery)
                (type fixnum nx nbmin nb lwkopt ldwork k iws iinfo
                           ib i))
      (setf info 0)
      (setf nb (ilaenv 1 "DGELQF" " " m n -1 -1))
      (setf lwkopt (f2cl-lib:int-mul m nb))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)

```

```

        (coerce (the fixnum lwkopt) 'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((< m 0)
   (setf info -1))
  ((< n 0)
   (setf info -2))
  ((< lda (max (the fixnum 1) (the fixnum m)))
   (setf info -4))
  ((and
    (< lwork (max (the fixnum 1) (the fixnum m)))
    (not lquery))
   (setf info -7)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DGELQF" (f2cl-lib:int-sub info))
   (go end_label))
  (lquery
   (go end_label)))
(setf k (min (the fixnum m) (the fixnum n)))
(cond
  ((= k 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nbmin 2)
(setf nx 0)
(setf iws m)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
        (max (the fixnum 0)
              (the fixnum
               (ilaenv 3 "DGELQF" " " m n -1 -1)))))
  (cond
   ((< nx k)
    (setf ldwork m)
    (setf iws (f2cl-lib:int-mul ldwork nb))
    (cond
     ((< lwork iws)
      (setf nb (the fixnum (truncate lwork ldwork)))
      (setf nbmin
            (max (the fixnum 2)
                  (the fixnum
                   (ilaenv 2 "DGELQF" " " m n -1 -1))))))))))
  (cond
   ((and (>= nb nbmin) (< nb k) (< nx k))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))

```

```

                                ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub nx))) nil)
(tagbody
  (setf ib
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))
      (the fixnum nb)))
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
      (dgelq2 ib (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
        (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
        lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
        iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
      (setf iinfo var-6))
    (cond
      ((<= (f2cl-lib:int-add i ib) m)
        (dlarft "Forward" "Rowwise"
          (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
          (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
          lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
          ldwork)
        (dlarfb "Right" "No transpose" "Forward" "Rowwise"
          (f2cl-lib:int-add (f2cl-lib:int-sub m i ib) 1)
          (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
          (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
          lda work ldwork
          (f2cl-lib:array-slice a
            double-float
            ((+ i ib) i)
            ((1 lda) (1 *)))
          lda
          (f2cl-lib:array-slice work double-float ((+ ib 1) ((1 *)))
            ldwork))))))
    (t
      (setf i 1)))
  (if (<= i k)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
      (dgelq2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
        (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
        (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
        (f2cl-lib:array-slice tau double-float (i) ((1 *))) work iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
      (setf iinfo var-6))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum iws) 'double-float)))
end_label
  (return (values nil nil nil nil nil nil nil info))))

```

dgeqr2 LAPACK

— dgeqr2.input —

```

)set break resume
)sys rm -f dgeqr2.output
)spool dgeqr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgeqr2.help —

```

=====
dgeqr2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DGEQR2 - a QR factorization of a real m by n matrix A

SYNOPSIS

SUBROUTINE DGEQR2(M, N, A, LDA, TAU, WORK, INFO)

INTEGER INFO, LDA, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

PURPOSE

DGEQR2 computes a QR factorization of a real m by n matrix A: $A = Q * R$.

ARGUMENTS

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the m by n matrix A. On exit, the elements on and above the diagonal of the array contain the min(m,n) by n upper trapezoidal matrix R (R is upper triangular if $m \geq n$); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors (see Further Details). LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,M)$.

TAU (output) DOUBLE PRECISION array, dimension (min(M,N))
 The scalar factors of the elementary reflectors (see Further Details).

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

FURTHER DETAILS

The matrix Q is represented as a product of elementary reflectors

$$Q = H(1) H(2) \dots H(k), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with
 $v(1:i-1) = 0$ and $v(i) = 1$; $v(i+1:m)$ is stored on exit in $A(i+1:m,i)$,
 and tau in TAU(i).

— dgeqr2.f —

```

SUBROUTINE DGEQR2( M, N, A, LDA, TAU, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
*  INTEGER          INFO, LDA, M, N
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )

```

```

*      ..
*
*      =====
*
*      .. Parameters ..
      DOUBLE PRECISION    ONE
      PARAMETER            ( ONE = 1.0D+0 )
*
*      ..
*      .. Local Scalars ..
      INTEGER              I, K
      DOUBLE PRECISION     AII
*
*      ..
*      .. External Subroutines ..
      EXTERNAL             DLARF, DLARFG, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC             MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      IF( M.LT.0 ) THEN
        INFO = -1
      ELSE IF( N.LT.0 ) THEN
        INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
        INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DGEQR2', -INFO )
        RETURN
      END IF
*
      K = MIN( M, N )
*
      DO 10 I = 1, K
*
*        Generate elementary reflector H(i) to annihilate A(i+1:m,i)
*
        CALL DLARFG( M-I+1, A( I, I ), A( MIN( I+1, M ), I ), 1,
$          TAU( I ) )
        IF( I.LT.N ) THEN
*
*          Apply H(i) to A(i:m,i+1:n) from the left
*
          AII = A( I, I )
          A( I, I ) = ONE
          CALL DLARF( 'Left', M-I+1, N-I, A( I, I ), 1, TAU( I ),

```



```

$          A( I, I+1 ), LDA, WORK )
      A( I, I ) = AII
    END IF
10 CONTINUE
    RETURN
*
*      End of DGEQR2
*
    END

```

— LAPACK dgeqr2 —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgeqr2 (m n a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
      (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (k 0))
        (declare (type (double-float) aii) (type fixnum i k))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info -1))
          ((< n 0)
            (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info -4)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DGEQR2" (f2cl-lib:int-sub info))
            (go end_label)))
        (setf k (min (the fixnum m) (the fixnum n)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i k) nil)
          (tagbody
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
                (f2cl-lib:array-slice a
                  double-float

```

```

((min (f2cl-lib:int-add i 1) m) i)
((1 lda) (1 *)))
1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
var-1)
(setf (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%) var-4))
(cond
((< i n)
(setf aii
(f2cl-lib:fref a-%data%
(i i)
((1 lda) (1 *))
a-%offset%))
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
one)
(dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
(f2cl-lib:int-sub n i)
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
(f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
(f2cl-lib:array-slice a
double-float
(i (f2cl-lib:int-add i 1))
((1 lda) (1 *)))
lda work)
(setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
aii))))))
end_label
(return (values nil nil nil nil nil nil info))))))

```

dgeqrf LAPACK

— dgeqrf.input —

```

)set break resume
)sys rm -f dgeqrf.output
)spool dgeqrf.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgeqrf.help —

=====

dgeqrf examples

=====

=====

Man Page Details

=====

NAME

DGEQRF - a QR factorization of a real M-by-N matrix A

SYNOPSIS

SUBROUTINE DGEQRF(M, N, A, LDA, TAU, WORK, LWORK, INFO)

INTEGER INFO, LDA, LWORK, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

PURPOSE

DGEQRF computes a QR factorization of a real M-by-N matrix A: $A = Q * R$.

ARGUMENTS

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N matrix A. On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if $m \geq n$); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors (see Further Details).

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

TAU (output) DOUBLE PRECISION array, dimension (min(M,N))
The scalar factors of the elementary reflectors (see Further Details).

WORK (workspace/output) DOUBLE PRECISION array, dimension
(MAX(1,LWORK))

On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER

The dimension of the array WORK. LWORK $\geq \max(1,N)$. For optimum performance LWORK $\geq N \cdot \text{NB}$, where NB is the optimal block-size.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

FURTHER DETAILS

The matrix Q is represented as a product of elementary reflectors

$$Q = H(1) H(2) \dots H(k), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with
v(1:i-1) = 0 and v(i) = 1; v(i+1:m) is stored on exit in A(i+1:m,i),
and tau in TAU(i).

—————

— dgeqrf.f —

SUBROUTINE DGEQRF(M, N, A, LDA, TAU, WORK, LWORK, INFO)

```
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
*  INTEGER          INFO, LDA, LWORK, M, N
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*  ..
```

```

*
* =====
*
* .. Local Scalars ..
LOGICAL          LQUERY
INTEGER          I, IB, IINFO, IWS, K, LDWORK, LWKOPT, NB,
$               NBMIN, NX
*
* ..
* .. External Subroutines ..
EXTERNAL         DGEQR2, DLARFB, DLARFT, XERBLA
*
* ..
* .. Intrinsic Functions ..
INTRINSIC        MAX, MIN
*
* ..
* .. External Functions ..
INTEGER          ILAENV
EXTERNAL         ILAENV
*
* ..
* .. Executable Statements ..
*
* Test the input arguments
*
INFO = 0
NB = ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
LWKOPT = N*NB
WORK( 1 ) = LWKOPT
LQUERY = ( LWORK.EQ.-1 )
IF( M.LT.0 ) THEN
    INFO = -1
ELSE IF( N.LT.0 ) THEN
    INFO = -2
ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
    INFO = -4
ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
    INFO = -7
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DGEQRF', -INFO )
    RETURN
ELSE IF( LQUERY ) THEN
    RETURN
END IF
*
* Quick return if possible
*
K = MIN( M, N )
IF( K.EQ.0 ) THEN
    WORK( 1 ) = 1
    RETURN
END IF

```

```

*
NBMIN = 2
NX = 0
IWS = N
IF( NB.GT.1 .AND. NB.LT.K ) THEN
*
*   Determine when to cross over from blocked to unblocked code.
*
NX = MAX( 0, ILAENV( 3, 'DGEQRF', ' ', M, N, -1, -1 ) )
IF( NX.LT.K ) THEN
*
*   Determine if workspace is large enough for blocked code.
*
LDWORK = N
IWS = LDWORK*N
IF( LWORK.LT.IWS ) THEN
*
*   Not enough workspace to use optimal NB:  reduce NB and
*   determine the minimum value of NB.
*
NB = LWORK / LDWORK
NBMIN = MAX( 2, ILAENV( 2, 'DGEQRF', ' ', M, N, -1,
$      -1 ) )
      END IF
      END IF
      END IF
*
IF( NB.GE.NBMIN .AND. NB.LT.K .AND. NX.LT.K ) THEN
*
*   Use blocked code initially
*
DO 10 I = 1, K - NX, NB
  IB = MIN( K-I+1, NB )
*
*   Compute the QR factorization of the current block
*   A(i:m,i:i+ib-1)
*
  CALL DGEQR2( M-I+1, IB, A( I, I ), LDA, TAU( I ), WORK,
$             IINFO )
  IF( I+IB.LE.N ) THEN
*
*   Form the triangular factor of the block reflector
*   H = H(i) H(i+1) . . . H(i+ib-1)
*
  CALL DLARFT( 'Forward', 'Columnwise', M-I+1, IB,
$             A( I, I ), LDA, TAU( I ), WORK, LDWORK )
*
*   Apply H' to A(i:m,i+ib:n) from the left
*
  CALL DLARFB( 'Left', 'Transpose', 'Forward',

```

```

$           'Columnwise', M-I+1, N-I-IB+1, IB,
$           A( I, I ), LDA, WORK, LDWORK, A( I, I+IB ),
$           LDA, WORK( IB+1 ), LDWORK )
      END IF
10    CONTINUE
      ELSE
        I = 1
      END IF
*
*      Use unblocked code to factor the last or only block.
*
      IF( I.LE.K )
$       CALL DGEQR2( M-I+1, N-I+1, A( I, I ), LDA, TAU( I ), WORK,
$                   IINFO )
*
      WORK( 1 ) = IWS
      RETURN
*
*      End of DGEQRF
*
      END

```

— LAPACK dgeqrf —

```

(defun dgeqrf (m n a lda tau work lwork info)
  (declare (type (simple-array double-float (*)) work tau a)
    (type fixnum info lwork lda n m))
  (f2cl-lib:with-multi-array-data
    ((a double-float a-%data% a-%offset%)
     (tau double-float tau-%data% tau-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (k 0) (ldwork 0) (lwkopt 0) (nb 0)
      (nbmin 0) (nx 0) (lquery nil))
      (declare (type (member t nil) lquery)
        (type fixnum nx nbmin nb lwkopt ldwork k iws iinfo
          ib i))
      (setf info 0)
      (setf nb (ilaenv 1 "DGEQRF" " " m n -1 -1))
      (setf lwkopt (f2cl-lib:int-mul n nb))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum lwkopt) 'double-float))
      (setf lquery (coerce (= lwork -1) '(member t nil)))
      (cond
        ((< m 0)
          (setf info -1))
        ((< n 0)

```

```

      (setf info -2))
    ((< lda (max (the fixnum 1) (the fixnum m)))
      (setf info -4))
    ((and
      (< lwork (max (the fixnum 1) (the fixnum n)))
      (not lquery))
      (setf info -7)))
  (cond
    ((/= info 0)
      (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DGEQRF" (f2cl-lib:int-sub info))
      (go end_label))
    (lquery
      (go end_label)))
  (setf k (min (the fixnum m) (the fixnum n)))
  (cond
    ((= k 0)
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum 1) 'double-float))
      (go end_label)))
  (setf nbmin 2)
  (setf nx 0)
  (setf iws n)
  (cond
    ((and (> nb 1) (< nb k))
      (setf nx
        (max (the fixnum 0)
          (the fixnum
            (ilaenv 3 "DGEQRF" " " " m n -1 -1)))))
    (cond
      ((< nx k)
        (setf ldwork n)
        (setf iws (f2cl-lib:int-mul ldwork nb))
        (cond
          ((< lwork iws)
            (setf nb (the fixnum (truncate lwork ldwork)))
            (setf nbmin
              (max (the fixnum 2)
                (the fixnum
                  (ilaenv 2 "DGEQRF" " " " m n -1 -1))))))))))
  (cond
    ((and (>= nb nbmin) (< nb k) (< nx k))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i nb))
        ((> i (f2cl-lib:int-add k (f2cl-lib:int-sub nx))) nil)
        (tagbody
          (setf ib
            (min
              (the fixnum
                (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))

```



```

        (the fixnum nb)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (dgeqr2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
    iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
  (setf iinfo var-6))
(cond
  ((<= (f2cl-lib:int-add i ib) n)
   (dlarft "Forward" "Columnwise"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
    ldwork)
   (dlarfb "Left" "Transpose" "Forward" "Columnwise"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    lda work ldwork
    (f2cl-lib:array-slice a
      double-float
      (i (f2cl-lib:int-add i ib))
      ((1 lda) (1 *)))
    lda
    (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
    ldwork))))))
(t
  (setf i 1)))
(if (<= i k)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (dgeqr2 (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
      (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
      (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
      (f2cl-lib:array-slice tau double-float (i) ((1 *))) work iinfo)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
    (setf iinfo var-6))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (coerce (the fixnum iws) 'double-float)))
end_label
(return (values nil nil nil nil nil nil nil info))))

```

dgesdd LAPACK

— dgesdd.input —

```

)set break resume
)sys rm -f dgesdd.output
)spool dgesdd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgesdd.help —

```

=====
dgesdd examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DGESDD - the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and right singular vectors

SYNOPSIS

```

SUBROUTINE DGESDD( JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK,
                  LWORK, IWORK, INFO )

```

| | |
|-----------|--|
| CHARACTER | JOBZ |
| INTEGER | INFO, LDA, LDU, LDVT, LWORK, M, N |
| INTEGER | IWORK(*) |
| DOUBLE | PRECISION A(LDA, *), S(*), U(LDU, *), VT(LDVT, *), WORK(*) |

PURPOSE

DGESDD computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm.

The SVD is written

$$A = U * \text{SIGMA} * \text{transpose}(V)$$

where SIGMA is an M-by-N matrix which is zero except for its $\min(m,n)$ diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first $\min(m,n)$ columns of U and V are the left and right singular vectors of A.

Note that the routine returns $VT = V^{**T}$, not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

ARGUMENTS

JOBZ (input) CHARACTER*1
Specifies options for computing all or part of the matrix U:
= 'A': all M columns of U and all N rows of V^{**T} are returned in the arrays U and VT; = 'S': the first $\min(M,N)$ columns of U and the first $\min(M,N)$ rows of V^{**T} are returned in the arrays U and VT; = 'O': If $M \geq N$, the first N columns of U are overwritten on the array A and all rows of V^{**T} are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V^{**T} are overwritten in the array A; = 'N': no columns of U or rows of V^{**T} are computed.

M (input) INTEGER
The number of rows of the input matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the input matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N matrix A. On exit, if $JOBZ = 'O'$, A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if $M \geq N$; A is overwritten with the first M rows of V^{**T} (the right singular vectors, stored rowwise) otherwise. if $JOBZ \neq 'O'$, the contents of A are destroyed.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

S (output) DOUBLE PRECISION array, dimension (min(M,N))
The singular values of A, sorted so that $S(i) \geq S(i+1)$.

U (output) DOUBLE PRECISION array, dimension (LDU,UCOL)
 UCOL = M if JOBZ = 'A' or JOBZ = 'O' and $M < N$; UCOL = min(M,N) if JOBZ = 'S'. If JOBZ = 'A' or JOBZ = 'O' and $M < N$, U contains the M-by-M orthogonal matrix U; if JOBZ = 'S', U contains the first min(M,N) columns of U (the left singular vectors, stored columnwise); if JOBZ = 'O' and $M \geq N$, or JOBZ = 'N', U is not referenced.

LDU (input) INTEGER
 The leading dimension of the array U. $LDU \geq 1$; if JOBZ = 'S' or 'A' or JOBZ = 'O' and $M < N$, $LDU \geq M$.

VT (output) DOUBLE PRECISION array, dimension (LDVT,N)
 If JOBZ = 'A' or JOBZ = 'O' and $M \geq N$, VT contains the N-by-N orthogonal matrix V^*T ; if JOBZ = 'S', VT contains the first min(M,N) rows of V^*T (the right singular vectors, stored rowwise); if JOBZ = 'O' and $M < N$, or JOBZ = 'N', VT is not referenced.

LDVT (input) INTEGER
 The leading dimension of the array VT. $LDVT \geq 1$; if JOBZ = 'A' or JOBZ = 'O' and $M \geq N$, $LDVT \geq N$; if JOBZ = 'S', $LDVT \geq \min(M,N)$.

WORK (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
 On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

LWORK (input) INTEGER
 The dimension of the array WORK. $LWORK \geq 1$. If JOBZ = 'N', $LWORK \geq 3 \cdot \min(M,N) + \max(\max(M,N), 7 \cdot \min(M,N))$. If JOBZ = 'O', $LWORK \geq 3 \cdot \min(M,N) \cdot \min(M,N) + \max(\max(M,N), 5 \cdot \min(M,N) \cdot \min(M,N) + 4 \cdot \min(M,N))$. If JOBZ = 'S' or 'A', $LWORK \geq 3 \cdot \min(M,N) \cdot \min(M,N) + \max(\max(M,N), 4 \cdot \min(M,N) \cdot \min(M,N) + 4 \cdot \min(M,N))$. For good performance, LWORK should generally be larger. If LWORK = -1 but other input arguments are legal, WORK(1) returns the optimal LWORK.

IWORK (workspace) INTEGER array, dimension (8*min(M,N))

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: DBDSDC did not converge, updating process failed.

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of
California at Berkeley, USA

— dgesdd.f —

```

SUBROUTINE DGESDD( JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK,
$                LWORK, IWORK, INFO )
*
*  -- LAPACK driver routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
CHARACTER          JOBZ
INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
*
*  ..
*  .. Array Arguments ..
INTEGER            IWORK( * )
DOUBLE PRECISION   A( LDA, * ), S( * ), U( LDU, * ),
$                 VT( LDVT, * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION   ZERO, ONE
PARAMETER          ( ZERO = 0.0D0, ONE = 1.0D0 )
*
*  ..
*  .. Local Scalars ..
LOGICAL            LQUERY, WNTQA, WNTQAS, WNTQN, WNTQO, WNTQS
INTEGER            BDSPAC, BLK, CHUNK, I, IE, IERR, IL,
$                 IR, ISCL, ITAU, ITAUP, ITAUQ, IU, IVT, LDWKVT,
$                 LDWRKL, LDWRKR, LDWRKU, MAXWRK, MINMN, MINWRK,
$                 MNTHR, NWORK, WRKBL
DOUBLE PRECISION   ANRM, BIGNUM, EPS, SMLNUM
*
*  ..
*  .. Local Arrays ..
INTEGER            IDUM( 1 )
DOUBLE PRECISION   DUM( 1 )
*
*  ..
*  .. External Subroutines ..
EXTERNAL           DBDSDC, DGEBRD, DGELQF, DGEMM, DGEQRF, DLACPY,
$                 DLASCL, DLASET, DORGBR, DORGLQ, DORGQR, DORMBR,
$                 XERBLA
*
*  ..

```

```

*      .. External Functions ..
LOGICAL          LSAME
INTEGER          ILAENV
DOUBLE PRECISION DLAMCH, DLANGE
EXTERNAL         DLAMCH, DLANGE, ILAENV, LSAME
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC         DBLE, INT, MAX, MIN, SQRT
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      MINMN = MIN( M, N )
      MNTHR = INT( MINMN*11.0D0 / 6.0D0 )
      WNTQA = LSAME( JOBZ, 'A' )
      WNTQS = LSAME( JOBZ, 'S' )
      WNTQAS = WNTQA .OR. WNTQS
      WNTQO = LSAME( JOBZ, 'O' )
      WNTQN = LSAME( JOBZ, 'N' )
      MINWRK = 1
      MAXWRK = 1
      LQUERY = ( LWORK.EQ.-1 )
*
      IF( .NOT.( WNTQA .OR. WNTQS .OR. WNTQO .OR. WNTQN ) ) THEN
          INFO = -1
      ELSE IF( M.LT.0 ) THEN
          INFO = -2
      ELSE IF( N.LT.0 ) THEN
          INFO = -3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
          INFO = -5
      ELSE IF( LDU.LT.1 .OR. ( WNTQAS .AND. LDU.LT.M ) .OR.
$          ( WNTQO .AND. M.LT.N .AND. LDU.LT.M ) ) THEN
          INFO = -8
      ELSE IF( LDVT.LT.1 .OR. ( WNTQA .AND. LDVT.LT.N ) .OR.
$          ( WNTQS .AND. LDVT.LT.MINMN ) .OR.
$          ( WNTQO .AND. M.GE.N .AND. LDVT.LT.N ) ) THEN
          INFO = -10
      END IF
*
*      Compute workspace
*      (Note: Comments in the code beginning "Workspace:" describe the
*      minimal amount of workspace needed at that point in the code,
*      as well as the preferred amount for good performance.
*      NB refers to the optimal block size for the immediately
*      following subroutine, as returned by ILAENV.)
*
      IF( INFO.EQ.0 .AND. M.GT.0 .AND. N.GT.0 ) THEN

```

```

      IF( M.GE.N ) THEN
*
*       Compute space needed for DBDSDC
*
      IF( WNTQN ) THEN
        BDSPAC = 7*N
      ELSE
        BDSPAC = 3*N*N + 4*N
      END IF
      IF( M.GE.MNTHR ) THEN
        IF( WNTQN ) THEN
*
*       Path 1 (M much larger than N, JOBZ='N')
*
          WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1,
$             -1 )
          WRKBL = MAX( WRKBL, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          MAXWRK = MAX( WRKBL, BDSPAC+N )
          MINWRK = BDSPAC + N
        ELSE IF( WNTQO ) THEN
*
*       Path 2 (M much larger than N, JOBZ='O')
*
          WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
          WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$             N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORMBR', 'QLN', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, BDSPAC+3*N )
          MAXWRK = WRKBL + 2*N*N
          MINWRK = BDSPAC + 2*N*N + 3*N
        ELSE IF( WNTQS ) THEN
*
*       Path 3 (M much larger than N, JOBZ='S')
*
          WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
          WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$             N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORMBR', 'QLN', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, BDSPAC+3*N )

```

```

MAXWRK = WRKBL + N*N
MINWRK = BDSPAC + N*N + 3*N
ELSE IF( WNTQA ) THEN
*
*       Path 4 (M much larger than N, JOBZ='A')
*
WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
WRKBL = MAX( WRKBL, N+M*ILAENV( 1, 'DORGQR', ' ', M,
$       M, N, -1 ) )
WRKBL = MAX( WRKBL, 3*N+2*N*
$       ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'QLN', N, N, N, -1 ) )
WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
WRKBL = MAX( WRKBL, BDSPAC+3*N )
MAXWRK = WRKBL + N*N
MINWRK = BDSPAC + N*N + 3*N
END IF
ELSE
*
*       Path 5 (M at least N, but not much larger)
*
WRKBL = 3*N + ( M+N )*ILAENV( 1, 'DGEBRD', ' ', M, N, -1,
$       -1 )
IF( WNTQN ) THEN
    MAXWRK = MAX( WRKBL, BDSPAC+3*N )
    MINWRK = 3*N + MAX( M, BDSPAC )
ELSE IF( WNTQO ) THEN
    WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'QLN', M, N, N, -1 ) )
    WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
    WRKBL = MAX( WRKBL, BDSPAC+3*N )
    MAXWRK = WRKBL + M*N
    MINWRK = 3*N + MAX( M, N*N+BDSPAC )
ELSE IF( WNTQS ) THEN
    WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'QLN', M, N, N, -1 ) )
    WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
    MAXWRK = MAX( WRKBL, BDSPAC+3*N )
    MINWRK = 3*N + MAX( M, BDSPAC )
ELSE IF( WNTQA ) THEN
    WRKBL = MAX( WRKBL, 3*N+M*
$       ILAENV( 1, 'DORMBR', 'QLN', M, M, N, -1 ) )
    WRKBL = MAX( WRKBL, 3*N+N*
$       ILAENV( 1, 'DORMBR', 'PRT', N, N, N, -1 ) )
    MAXWRK = MAX( MAXWRK, BDSPAC+3*N )
    MINWRK = 3*N + MAX( M, BDSPAC )

```



```

        END IF
    END IF
ELSE
*
*       Compute space needed for DBDSDC
*
    IF( WNTQN ) THEN
        BDSPAC = 7*M
    ELSE
        BDSPAC = 3*M*M + 4*M
    END IF
    IF( N.GE.MNTHR ) THEN
        IF( WNTQN ) THEN
*
*       Path 1t (N much larger than M, JOBZ='N')
*
            WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1,
$              -1 )
            WRKBL = MAX( WRKBL, 3*M+2*M*
$              ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
            MAXWRK = MAX( WRKBL, BDSPAC+M )
            MINWRK = BDSPAC + M
        ELSE IF( WNTQO ) THEN
*
*       Path 2t (N much larger than M, JOBZ='O')
*
            WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
            WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$              N, M, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+2*M*
$              ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+M*
$              ILAENV( 1, 'DORMBR', 'QLN', M, M, M, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+M*
$              ILAENV( 1, 'DORMBR', 'PRT', M, M, M, -1 ) )
            WRKBL = MAX( WRKBL, BDSPAC+3*M )
            MAXWRK = WRKBL + 2*M*M
            MINWRK = BDSPAC + 2*M*M + 3*M
        ELSE IF( WNTQS ) THEN
*
*       Path 3t (N much larger than M, JOBZ='S')
*
            WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
            WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$              N, M, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+2*M*
$              ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+M*
$              ILAENV( 1, 'DORMBR', 'QLN', M, M, M, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+M*

```

```

$          ILAENV( 1, 'DORMBR', 'PRT', M, M, M, -1 ) )
WRKBL = MAX( WRKBL, BDSPAC+3*M )
MAXWRK = WRKBL + M*M
MINWRK = BDSPAC + M*M + 3*M
ELSE IF( WNTQA ) THEN
*
*      Path 4t (N much larger than M, JOBZ='A')
*
WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
WRKBL = MAX( WRKBL, M+N*ILAENV( 1, 'DORGLQ', ' ', N,
$          N, M, -1 ) )
WRKBL = MAX( WRKBL, 3*M+2*M*
$          ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'QLN', M, M, M, -1 ) )
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'PRT', M, M, M, -1 ) )
WRKBL = MAX( WRKBL, BDSPAC+3*M )
MAXWRK = WRKBL + M*M
MINWRK = BDSPAC + M*M + 3*M
END IF
ELSE
*
*      Path 5t (N greater than M, but not much larger)
*
WRKBL = 3*M + ( M+N )*ILAENV( 1, 'DGEBRD', ' ', M, N, -1,
$          -1 )
IF( WNTQN ) THEN
MAXWRK = MAX( WRKBL, BDSPAC+3*M )
MINWRK = 3*M + MAX( N, BDSPAC )
ELSE IF( WNTQO ) THEN
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'QLN', M, M, N, -1 ) )
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'PRT', M, N, M, -1 ) )
WRKBL = MAX( WRKBL, BDSPAC+3*M )
MAXWRK = WRKBL + M*N
MINWRK = 3*M + MAX( N, M*M+BDSPAC )
ELSE IF( WNTQS ) THEN
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'QLN', M, M, N, -1 ) )
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'PRT', M, N, M, -1 ) )
MAXWRK = MAX( WRKBL, BDSPAC+3*M )
MINWRK = 3*M + MAX( N, BDSPAC )
ELSE IF( WNTQA ) THEN
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'QLN', M, M, N, -1 ) )
WRKBL = MAX( WRKBL, 3*M+M*
$          ILAENV( 1, 'DORMBR', 'PRT', N, N, M, -1 ) )

```

```

        MAXWRK = MAX( WRKBL, BDSPAC+3*M )
        MINWRK = 3*M + MAX( N, BDSPAC )
    END IF
    END IF
    END IF
    WORK( 1 ) = MAXWRK
END IF

*
IF( LWORK.LT.MINWRK .AND. .NOT.LQUERY ) THEN
    INFO = -12
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DGESDD', -INFO )
    RETURN
ELSE IF( LQUERY ) THEN
    RETURN
END IF

*
* Quick return if possible
*
IF( M.EQ.0 .OR. N.EQ.0 ) THEN
    IF( LWORK.GE.1 )
$       WORK( 1 ) = ONE
    RETURN
END IF

*
* Get machine constants
*
EPS = DLAMCH( 'P' )
SMLNUM = SQRT( DLAMCH( 'S' ) ) / EPS
BIGNUM = ONE / SMLNUM

*
* Scale A if max element outside range [SMLNUM,BIGNUM]
*
ANRM = DLANGE( 'M', M, N, A, LDA, DUM )
ISCL = 0
IF( ANRM.GT.ZERO .AND. ANRM.LT.SMLNUM ) THEN
    ISCL = 1
    CALL DLASCL( 'G', 0, 0, ANRM, SMLNUM, M, N, A, LDA, IERR )
ELSE IF( ANRM.GT.BIGNUM ) THEN
    ISCL = 1
    CALL DLASCL( 'G', 0, 0, ANRM, BIGNUM, M, N, A, LDA, IERR )
END IF

*
IF( M.GE.N ) THEN

*
* A has at least as many rows as columns. If A has sufficiently
* more rows than columns, first reduce using the QR
* decomposition (if sufficient workspace available)
*

```

```

      IF( M.GE.MNTHR ) THEN
*
*       IF( WNTQN ) THEN
*
*           Path 1 (M much larger than N, JOBZ='N')
*           No singular vectors to be computed
*
*           ITAU = 1
*           NWORK = ITAU + N
*
*           Compute A=Q*R
*           (Workspace: need 2*N, prefer N+N*NB)
*
*           CALL DGEQRF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$               LWORK-NWORK+1, IERR )
*
*           Zero out below R
*
*           CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ), LDA )
*           IE = 1
*           ITAUQ = IE + N
*           ITAUP = ITAUQ + N
*           NWORK = ITAUP + N
*
*           Bidiagonalize R in A
*           (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
*           CALL DGEBRD( N, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$               WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$               IERR )
*           NWORK = IE + N
*
*           Perform bidiagonal SVD, computing singular values only
*           (Workspace: need N+BDSPAC)
*
*           CALL DBDSDC( 'U', 'N', N, S, WORK( IE ), DUM, 1, DUM, 1,
$               DUM, IDUM, WORK( NWORK ), IWORK, INFO )
*
*       ELSE IF( WNTQO ) THEN
*
*           Path 2 (M much larger than N, JOBZ = 'O')
*           N left singular vectors to be overwritten on A and
*           N right singular vectors to be computed in VT
*
*           IR = 1
*
*           WORK(IR) is LDWRKR by N
*
*           IF( LWORK.GE.LDA*N+N*N+3*N+BDSPAC ) THEN
*               LDWRKR = LDA

```

```

ELSE
    LDWRKR = ( LWORK-N*N-3*N-BDSPAC ) / N
END IF
ITAU = IR + LDWRKR*N
NWORK = ITAU + N
*
*   Compute A=Q*R
*   (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
CALL DGEQRF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$           LWORK-NWORK+1, IERR )
*
*   Copy R to WORK(IR), zeroing out below it
*
CALL DLACPY( 'U', N, N, A, LDA, WORK( IR ), LDWRKR )
CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, WORK( IR+1 ),
$           LDWRKR )
*
*   Generate Q in A
*   (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$           WORK( NWORK ), LWORK-NWORK+1, IERR )
IE = ITAU
ITAUQ = IE + N
ITAUP = ITAUQ + N
NWORK = ITAUP + N
*
*   Bidiagonalize R in VT, copying result to WORK(IR)
*   (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
CALL DGEBRD( N, N, WORK( IR ), LDWRKR, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ), WORK( NWORK ),
$           LWORK-NWORK+1, IERR )
*
*   WORK(IU) is N by N
*
IU = NWORK
NWORK = IU + N*N
*
*   Perform bidiagonal SVD, computing left singular vectors
*   of bidiagonal matrix in WORK(IU) and computing right
*   singular vectors of bidiagonal matrix in VT
*   (Workspace: need N+N*N+BDSPAC)
*
CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), WORK( IU ), N,
$           VT, LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$           INFO )
*
*   Overwrite WORK(IU) by left singular vectors of R

```

```

*      and VT by right singular vectors of R
*      (Workspace: need 2*N*N+3*N, prefer 2*N*N+2*N+N*NB)
*
*      CALL DORMBR( 'Q', 'L', 'N', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUQ ), WORK( IU ), N, WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*      CALL DORMBR( 'P', 'R', 'T', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*      Multiply Q in A by left singular vectors of R in
*      WORK(IU), storing result in WORK(IR) and copying to A
*      (Workspace: need 2*N*N, prefer N*N+M*N)
*
*      DO 10 I = 1, M, LDWRKR
*          CHUNK = MIN( M-I+1, LDWRKR )
*          CALL DGEMM( 'N', 'N', CHUNK, N, N, ONE, A( I, 1 ),
$              LDA, WORK( IU ), N, ZERO, WORK( IR ),
$              LDWRKR )
*          CALL DLACPY( 'F', CHUNK, N, WORK( IR ), LDWRKR,
$              A( I, 1 ), LDA )
10      CONTINUE
*
*      ELSE IF( WNTQS ) THEN
*
*          Path 3 (M much larger than N, JOBZ='S')
*          N left singular vectors to be computed in U and
*          N right singular vectors to be computed in VT
*
*          IR = 1
*
*          WORK(IR) is N by N
*
*          LDWRKR = N
*          ITAU = IR + LDWRKR*N
*          NWORK = ITAU + N
*
*          Compute A=Q*R
*          (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
*          CALL DGEQRF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*          Copy R to WORK(IR), zeroing out below it
*
*          CALL DLACPY( 'U', N, N, A, LDA, WORK( IR ), LDWRKR )
*          CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, WORK( IR+1 ),
$              LDWRKR )
*
*          Generate Q in A

```

```

*          (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*          CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$              WORK( NWORK ), LWORK-NWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          NWORK = ITAUP + N
*
*          Bidiagonalize R in WORK(IR)
*          (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
*          CALL DGEBRD( N, N, WORK( IR ), LDWRKR, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ), WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*          Perform bidiagonal SVD, computing left singular vectors
*          of bidiagonal matrix in U and computing right singular
*          vectors of bidiagonal matrix in VT
*          (Workspace: need N+BDSPAC)
*
*          CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), U, LDU, VT,
$              LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$              INFO )
*
*          Overwrite U by left singular vectors of R and VT
*          by right singular vectors of R
*          (Workspace: need N*N+3*N, prefer N*N+2*N+N*NB)
*
*          CALL DORMBR( 'Q', 'L', 'N', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*          CALL DORMBR( 'P', 'R', 'T', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*          Multiply Q in A by left singular vectors of R in
*          WORK(IR), storing result in U
*          (Workspace: need N*N)
*
*          CALL DLACPY( 'F', N, N, U, LDU, WORK( IR ), LDWRKR )
*          CALL DGEMM( 'N', 'N', M, N, N, ONE, A, LDA, WORK( IR ),
$              LDWRKR, ZERO, U, LDU )
*
*          ELSE IF( WNTQA ) THEN
*
*          Path 4 (M much larger than N, JOBZ='A')
*          M left singular vectors to be computed in U and
*          N right singular vectors to be computed in VT

```

```

*
*      IU = 1
*
*      WORK(IU) is N by N
*
*      LDWRKU = N
*      ITAU = IU + LDWRKU*N
*      NWORK = ITAU + N
*
*      Compute A=Q*R, copying result to U
*      (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*      CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*      Generate Q in U
*      (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*      CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$              WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*      Produce R in A, zeroing out other entries
*
*      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ), LDA )
*      IE = ITAU
*      ITAUQ = IE + N
*      ITAUP = ITAUQ + N
*      NWORK = ITAUP + N
*
*      Bidiagonalize R in A
*      (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
*      CALL DGEBRD( N, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$              IERR )
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in WORK(IU) and computing right
*      singular vectors of bidiagonal matrix in VT
*      (Workspace: need N*N*NB+BDSPAC)
*
*      CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), WORK( IU ), N,
$              VT, LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$              INFO )
*
*      Overwrite WORK(IU) by left singular vectors of R and VT
*      by right singular vectors of R
*      (Workspace: need N*N+3*N, prefer N*N+2*N+N*NB)
*
*      CALL DORMBR( 'Q', 'L', 'N', N, N, N, A, LDA,

```



```

$          WORK( ITAUQ ), WORK( IU ), LDWRKU,
$          WORK( NWORK ), LWORK-NWORK+1, IERR )
      CALL DORMBR( 'P', 'R', 'T', N, N, N, A, LDA,
$          WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*
*      Multiply Q in U by left singular vectors of R in
*      WORK(IU), storing result in A
*      (Workspace: need N*N)
*
      CALL DGEMM( 'N', 'N', M, N, N, ONE, U, LDU, WORK( IU ),
$          LDWRKU, ZERO, A, LDA )
*
*      Copy left singular vectors of A from A to U
*
      CALL DLACPY( 'F', M, N, A, LDA, U, LDU )
*
      END IF
*
    ELSE
*
      M .LT. MNTHR
*
*      Path 5 (M at least N, but not much larger)
*      Reduce to bidiagonal form without QR decomposition
*
      IE = 1
      ITAUQ = IE + N
      ITAUP = ITAUQ + N
      NWORK = ITAUP + N
*
*      Bidiagonalize A
*      (Workspace: need 3*N+M, prefer 3*N+(M+N)*NB)
*
      CALL DGEBRD( M, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$          WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$          IERR )
      IF( WNTQN ) THEN
*
*      Perform bidiagonal SVD, only computing singular values
*      (Workspace: need N+BDSPAC)
*
      CALL DBDSDC( 'U', 'N', N, S, WORK( IE ), DUM, 1, DUM, 1,
$          DUM, IDUM, WORK( NWORK ), IWORK, INFO )
      ELSE IF( WNTQO ) THEN
        IU = NWORK
        IF( LWORK.GE.M*N+3*N+BDSPAC ) THEN
*
*      WORK( IU ) is M by N
*

```

```

LDWRKU = M
NWORK = IU + LDWRKU*N
CALL DLASET( 'F', M, N, ZERO, ZERO, WORK( IU ),
$           LDWRKU )
ELSE
*
*   WORK( IU ) is N by N
*
LDWRKU = N
NWORK = IU + LDWRKU*N
*
*   WORK(IR) is LDWRKR by N
*
IR = NWORK
LDWRKR = ( LWORK-N*N-3*N ) / N
END IF
NWORK = IU + LDWRKU*N
*
*   Perform bidiagonal SVD, computing left singular vectors
*   of bidiagonal matrix in WORK(IU) and computing right
*   singular vectors of bidiagonal matrix in VT
*   (Workspace: need N+N*N+BDSPAC)
*
CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), WORK( IU ),
$           LDWRKU, VT, LDVT, DUM, IDUM, WORK( NWORK ),
$           IWORK, INFO )
*
*   Overwrite VT by right singular vectors of A
*   (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
CALL DORMBR( 'P', 'R', 'T', N, N, N, A, LDA,
$           WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$           LWORK-NWORK+1, IERR )
*
IF( LWORK.GE.M*N+3*N+BDSPAC ) THEN
*
*   Overwrite WORK(IU) by left singular vectors of A
*   (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
CALL DORMBR( 'Q', 'L', 'N', M, N, N, A, LDA,
$           WORK( ITAUQ ), WORK( IU ), LDWRKU,
$           WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*   Copy left singular vectors of A from WORK(IU) to A
*
CALL DLACPY( 'F', M, N, WORK( IU ), LDWRKU, A, LDA )
ELSE
*
*   Generate Q in A
*   (Workspace: need N*N+2*N, prefer N*N+N*N*NB)

```

```

*
*      CALL DORGBR( 'Q', M, N, N, A, LDA, WORK( ITAUQ ),
$          WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*      Multiply Q in A by left singular vectors of
*      bidiagonal matrix in WORK(IU), storing result in
*      WORK(IR) and copying to A
*      (Workspace: need 2*N*N, prefer N*N+M*N)
*
      DO 20 I = 1, M, LDWRKR
          CHUNK = MIN( M-I+1, LDWRKR )
          CALL DGEMM( 'N', 'N', CHUNK, N, N, ONE, A( I, 1 ),
$              LDA, WORK( IU ), LDWRKU, ZERO,
$              WORK( IR ), LDWRKR )
          CALL DLACPY( 'F', CHUNK, N, WORK( IR ), LDWRKR,
$              A( I, 1 ), LDA )
20      CONTINUE
      END IF
*
      ELSE IF( WNTQS ) THEN
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U and computing right singular
*      vectors of bidiagonal matrix in VT
*      (Workspace: need N+BDSPAC)
*
      CALL DLASET( 'F', M, N, ZERO, ZERO, U, LDU )
      CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), U, LDU, VT,
$          LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$          INFO )
*
*      Overwrite U by left singular vectors of A and VT
*      by right singular vectors of A
*      (Workspace: need 3*N, prefer 2*N+N*NB)
*
      CALL DORMBR( 'Q', 'L', 'N', M, N, N, A, LDA,
$          WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
      CALL DORMBR( 'P', 'R', 'T', N, N, N, A, LDA,
$          WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
      ELSE IF( WNTQA ) THEN
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U and computing right singular
*      vectors of bidiagonal matrix in VT
*      (Workspace: need N+BDSPAC)
*
      CALL DLASET( 'F', M, M, ZERO, ZERO, U, LDU )
      CALL DBDSDC( 'U', 'I', N, S, WORK( IE ), U, LDU, VT,

```

```

$          LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$          INFO )
*
*          Set the right corner of U to identity matrix
*
*          CALL DLASET( 'F', M-N, M-N, ZERO, ONE, U( N+1, N+1 ),
$          LDU )
*
*          Overwrite U by left singular vectors of A and VT
*          by right singular vectors of A
*          (Workspace: need N*N+2*N+M, prefer N*N+2*N+M*NB)
*
*          CALL DORMBR( 'Q', 'L', 'N', M, M, N, A, LDA,
$          WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*          CALL DORMBR( 'P', 'R', 'T', N, N, M, A, LDA,
$          WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*          END IF
*
*          END IF
*
*          ELSE
*
*          A has more columns than rows. If A has sufficiently more
*          columns than rows, first reduce using the LQ decomposition (if
*          sufficient workspace available)
*
*          IF( N.GE.MNTHR ) THEN
*
*          IF( WNTQN ) THEN
*
*          Path 1t (N much larger than M, JOBZ='N')
*          No singular vectors to be computed
*
*          ITAU = 1
*          NWORK = ITAU + M
*
*          Compute A=L*Q
*          (Workspace: need 2*M, prefer M+M*NB)
*
*          CALL DGELQF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*
*          Zero out above L
*
*          CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ), LDA )
*          IE = 1
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M

```

```

        NWORK = ITAUP + M
*
*      Bidiagonalize L in A
*      (Workspace: need 4*M, prefer 3*M+2*M*Nb)
*
      CALL DGEBRD( M, M, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$              IERR )
      NWORK = IE + M
*
*      Perform bidiagonal SVD, computing singular values only
*      (Workspace: need M+BDSPAC)
*
      CALL DBDSDC( 'U', 'N', M, S, WORK( IE ), DUM, 1, DUM, 1,
$              DUM, IDUM, WORK( NWORK ), IWORK, INFO )
*
      ELSE IF( WNTQO ) THEN
*
*      Path 2t (N much larger than M, JOBZ='O')
*      M right singular vectors to be overwritten on A and
*      M left singular vectors to be computed in U
*
      IVT = 1
*
*      IVT is M by M
*
      IL = IVT + M*M
      IF( LWORK.GE.M*N+M*M+3*M+BDSPAC ) THEN
*
*      WORK(IL) is M by N
*
*      LDWRKL = M
*      CHUNK = N
      ELSE
*      LDWRKL = M
*      CHUNK = ( LWORK-M*M ) / M
      END IF
      ITAU = IL + LDWRKL*M
      NWORK = ITAU + M
*
*      Compute A=L*Q
*      (Workspace: need M*M+2*M, prefer M*M+M*M*Nb)
*
      CALL DGELQF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
*
*      Copy L to WORK(IL), zeroing about above it
*
      CALL DLACPY( 'L', M, M, A, LDA, WORK( IL ), LDWRKL )
      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,

```

```

$                                WORK( IL+LDWRKL ), LDWRKL )
*
*      Generate Q in A
*      (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$           WORK( NWORK ), LWORK-NWORK+1, IERR )
$
IE = ITAU
ITAUQ = IE + M
ITAUP = ITAUQ + M
NWORK = ITAUP + M
*
*      Bidiagonalize L in WORK(IL)
*      (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*Nb)
*
CALL DGEBRD( M, M, WORK( IL ), LDWRKL, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ), WORK( NWORK ),
$           LWORK-NWORK+1, IERR )
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U, and computing right singular
*      vectors of bidiagonal matrix in WORK(IVT)
*      (Workspace: need M*M+M+BDSPAC)
*
CALL DBDSDC( 'U', 'I', M, S, WORK( IE ), U, LDU,
$           WORK( IVT ), M, DUM, IDUM, WORK( NWORK ),
$           IWORK, INFO )
*
*      Overwrite U by left singular vectors of L and WORK(IVT)
*      by right singular vectors of L
*      (Workspace: need 2*M*M+3*M, prefer 2*M*M+2*M+M*Nb)
*
CALL DORMBR( 'Q', 'L', 'N', M, M, M, WORK( IL ), LDWRKL,
$           WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$           LWORK-NWORK+1, IERR )
CALL DORMBR( 'P', 'R', 'T', M, M, M, WORK( IL ), LDWRKL,
$           WORK( ITAUP ), WORK( IVT ), M,
$           WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*      Multiply right singular vectors of L in WORK(IVT) by Q
*      in A, storing result in WORK(IL) and copying to A
*      (Workspace: need 2*M*M, prefer M*M+M*N)
*
DO 30 I = 1, N, CHUNK
    BLK = MIN( N-I+1, CHUNK )
    CALL DGEMM( 'N', 'N', M, BLK, M, ONE, WORK( IVT ), M,
$           A( 1, I ), LDA, ZERO, WORK( IL ), LDWRKL )
    CALL DLACPY( 'F', M, BLK, WORK( IL ), LDWRKL,
$           A( 1, I ), LDA )
30      CONTINUE

```

```

*
*       ELSE IF( WNTQS ) THEN
*
*           Path 3t (N much larger than M, JOBZ='S')
*           M right singular vectors to be computed in VT and
*           M left singular vectors to be computed in U
*
*           IL = 1
*
*           WORK(IL) is M by M
*
*           LDWRKL = M
*           ITAU = IL + LDWRKL*M
*           NWORK = ITAU + M
*
*           Compute A=L*Q
*           (Workspace: need M*M+2*M, prefer M*M+M+M*NB)
*
*           CALL DGELQF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$               LWORK-NWORK+1, IERR )
*
*           Copy L to WORK(IL), zeroing out above it
*
*           CALL DLACPY( 'L', M, M, A, LDA, WORK( IL ), LDWRKL )
*           CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$               WORK( IL+LDWRKL ), LDWRKL )
*
*           Generate Q in A
*           (Workspace: need M*M+2*M, prefer M*M+M+M*NB)
*
*           CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$               WORK( NWORK ), LWORK-NWORK+1, IERR )
*           IE = ITAU
*           ITAUQ = IE + M
*           ITAUP = ITAUQ + M
*           NWORK = ITAUP + M
*
*           Bidiagonalize L in WORK(IU), copying result to U
*           (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
*           CALL DGEBRD( M, M, WORK( IL ), LDWRKL, S, WORK( IE ),
$               WORK( ITAUQ ), WORK( ITAUP ), WORK( NWORK ),
$               LWORK-NWORK+1, IERR )
*
*           Perform bidiagonal SVD, computing left singular vectors
*           of bidiagonal matrix in U and computing right singular
*           vectors of bidiagonal matrix in VT
*           (Workspace: need M+BDSPAC)
*
*           CALL DBDSDC( 'U', 'I', M, S, WORK( IE ), U, LDU, VT,

```

```

$          LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$          INFO )
*
*      Overwrite U by left singular vectors of L and VT
*      by right singular vectors of L
*      (Workspace: need M*M+3*M, prefer M*M+2*M+M*NB)
*
$          CALL DORMBR( 'Q', 'L', 'N', M, M, M, WORK( IL ), LDWRKL,
$                      WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$                      LWORK-NWORK+1, IERR )
$          CALL DORMBR( 'P', 'R', 'T', M, M, M, WORK( IL ), LDWRKL,
$                      WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$                      LWORK-NWORK+1, IERR )
*
*      Multiply right singular vectors of L in WORK(IL) by
*      Q in A, storing result in VT
*      (Workspace: need M*M)
*
$          CALL DLACPY( 'F', M, M, VT, LDVT, WORK( IL ), LDWRKL )
$          CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IL ), LDWRKL,
$                      A, LDA, ZERO, VT, LDVT )
*
*      ELSE IF( WNTQA ) THEN
*
*          Path 4t (N much larger than M, JOBZ='A')
*          N right singular vectors to be computed in VT and
*          M left singular vectors to be computed in U
*
*          IVT = 1
*
*          WORK(IVT) is M by M
*
*          LDWKVT = M
*          ITAU = IVT + LDWKVT*M
*          NWORK = ITAU + M
*
*          Compute A=L*Q, copying result to VT
*          (Workspace: need M*M+2*M, prefer M*M+M+M*NB)
*
$          CALL DGELQF( M, N, A, LDA, WORK( ITAU ), WORK( NWORK ),
$                      LWORK-NWORK+1, IERR )
$          CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*          Generate Q in VT
*          (Workspace: need M*M+2*M, prefer M*M+M+M*NB)
*
$          CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$                      WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*          Produce L in A, zeroing out other entries

```



```

*
      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ), LDA )
      IE = ITAU
      ITAUQ = IE + M
      ITAUP = ITAUQ + M
      NWORK = ITAUP + M
*
*      Bidiagonalize L in A
*      (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
      CALL DGEBRD( M, M, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$              IERR )
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U and computing right singular
*      vectors of bidiagonal matrix in WORK(IVT)
*      (Workspace: need M+M*M+BDSPAC)
*
      CALL DBDSDC( 'U', 'I', M, S, WORK( IE ), U, LDU,
$              WORK( IVT ), LDWKVT, DUM, IDUM,
$              WORK( NWORK ), IWORK, INFO )
*
*      Overwrite U by left singular vectors of L and WORK(IVT)
*      by right singular vectors of L
*      (Workspace: need M*M+3*M, prefer M*M+2*M+M*NB)
*
      CALL DORMBR( 'Q', 'L', 'N', M, M, M, A, LDA,
$              WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$              LWORK-NWORK+1, IERR )
      CALL DORMBR( 'P', 'R', 'T', M, M, M, A, LDA,
$              WORK( ITAUP ), WORK( IVT ), LDWKVT,
$              WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*      Multiply right singular vectors of L in WORK(IVT) by
*      Q in VT, storing result in A
*      (Workspace: need M*M)
*
      CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IVT ), LDWKVT,
$              VT, LDVT, ZERO, A, LDA )
*
*      Copy right singular vectors of A from A to VT
*
      CALL DLACPY( 'F', M, N, A, LDA, VT, LDVT )
*
      END IF
*
      ELSE
*
*      N .LT. MNTHR

```

```

*
*      Path 5t (N greater than M, but not much larger)
*      Reduce to bidiagonal form without LQ decomposition
*
      IE = 1
      ITAUQ = IE + M
      ITAUP = ITAUQ + M
      NWORK = ITAUP + M
*
*      Bidiagonalize A
*      (Workspace: need 3*M+N, prefer 3*M+(M+N)*NB)
*
      CALL DGEBRD( M, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( NWORK ), LWORK-NWORK+1,
$              IERR )
      IF( WNTQN ) THEN
*
*          Perform bidiagonal SVD, only computing singular values
*          (Workspace: need M+BDSPAC)
*
      CALL DBDSDC( 'L', 'N', M, S, WORK( IE ), DUM, 1, DUM, 1,
$              DUM, IDUM, WORK( NWORK ), IWORK, INFO )
      ELSE IF( WNTQO ) THEN
          LDWKVT = M
          IVT = NWORK
          IF( LWORK.GE.M*N+3*M+BDSPAC ) THEN
*
*              WORK( IVT ) is M by N
*
          CALL DLASET( 'F', M, N, ZERO, ZERO, WORK( IVT ),
$              LDWKVT )
          NWORK = IVT + LDWKVT*N
          ELSE
*
*              WORK( IVT ) is M by M
*
          NWORK = IVT + LDWKVT*M
          IL = NWORK
*
*              WORK(IL) is M by CHUNK
*
          CHUNK = ( LWORK-M*M-3*M ) / M
          END IF
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U and computing right singular
*      vectors of bidiagonal matrix in WORK(IVT)
*      (Workspace: need M*M+BDSPAC)
*
      CALL DBDSDC( 'L', 'I', M, S, WORK( IE ), U, LDU,

```

```

$          WORK( IVT ), LDWKVT, DUM, IDUM,
$          WORK( NWORK ), IWORK, INFO )
*
*      Overwrite U by left singular vectors of A
*      (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*      CALL DORMBR( 'Q', 'L', 'N', M, M, N, A, LDA,
$          WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*
*      IF( LWORK.GE.M*N+3*M+BDSPAC ) THEN
*
*          Overwrite WORK(IVT) by left singular vectors of A
*          (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*          CALL DORMBR( 'P', 'R', 'T', M, N, M, A, LDA,
$              WORK( ITAUP ), WORK( IVT ), LDWKVT,
$              WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*          Copy right singular vectors of A from WORK(IVT) to A
*
*          CALL DLACPY( 'F', M, N, WORK( IVT ), LDWKVT, A, LDA )
*      ELSE
*
*          Generate P**T in A
*          (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*          CALL DORGBR( 'P', M, N, M, A, LDA, WORK( ITAUP ),
$              WORK( NWORK ), LWORK-NWORK+1, IERR )
*
*          Multiply Q in A by right singular vectors of
*          bidiagonal matrix in WORK(IVT), storing result in
*          WORK(IL) and copying to A
*          (Workspace: need 2*M*M, prefer M*M+M*N)
*
*          DO 40 I = 1, N, CHUNK
*              BLK = MIN( N-I+1, CHUNK )
*              CALL DGEMM( 'N', 'N', M, BLK, M, ONE, WORK( IVT ),
$                  LDWKVT, A( 1, I ), LDA, ZERO,
$                  WORK( IL ), M )
*              CALL DLACPY( 'F', M, BLK, WORK( IL ), M, A( 1, I ),
$                  LDA )
40          CONTINUE
*      END IF
*      ELSE IF( WNTQS ) THEN
*
*          Perform bidiagonal SVD, computing left singular vectors
*          of bidiagonal matrix in U and computing right singular
*          vectors of bidiagonal matrix in VT
*          (Workspace: need M+BDSPAC)

```

```

*
*      CALL DLASET( 'F', M, N, ZERO, ZERO, VT, LDVT )
*      CALL DBDSDC( 'L', 'I', M, S, WORK( IE ), U, LDU, VT,
$          LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$          INFO )
*
*      Overwrite U by left singular vectors of A and VT
*      by right singular vectors of A
*      (Workspace: need 3*M, prefer 2*M+M*N)
*
*      CALL DORMBR( 'Q', 'L', 'N', M, M, N, A, LDA,
$          WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*      CALL DORMBR( 'P', 'R', 'T', M, N, M, A, LDA,
$          WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*      ELSE IF( WNTQA ) THEN
*
*      Perform bidiagonal SVD, computing left singular vectors
*      of bidiagonal matrix in U and computing right singular
*      vectors of bidiagonal matrix in VT
*      (Workspace: need M+BDSPAC)
*
*      CALL DLASET( 'F', N, N, ZERO, ZERO, VT, LDVT )
*      CALL DBDSDC( 'L', 'I', M, S, WORK( IE ), U, LDU, VT,
$          LDVT, DUM, IDUM, WORK( NWORK ), IWORK,
$          INFO )
*
*      Set the right corner of VT to identity matrix
*
*      CALL DLASET( 'F', N-M, N-M, ZERO, ONE, VT( M+1, M+1 ),
$          LDVT )
*
*      Overwrite U by left singular vectors of A and VT
*      by right singular vectors of A
*      (Workspace: need 2*M+N, prefer 2*M+N*N)
*
*      CALL DORMBR( 'Q', 'L', 'N', M, M, N, A, LDA,
$          WORK( ITAUQ ), U, LDU, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*      CALL DORMBR( 'P', 'R', 'T', N, N, M, A, LDA,
$          WORK( ITAUP ), VT, LDVT, WORK( NWORK ),
$          LWORK-NWORK+1, IERR )
*      END IF
*
*      END IF
*
*      END IF
*
*      Undo scaling if necessary

```

```

*
      IF( ISCL.EQ.1 ) THEN
        IF( ANRM.GT.BIGNUM )
$          CALL DLASCL( 'G', 0, 0, BIGNUM, ANRM, MINMN, 1, S, MINMN,
$                    IERR )
        IF( ANRM.LT.SMLNUM )
$          CALL DLASCL( 'G', 0, 0, SMLNUM, ANRM, MINMN, 1, S, MINMN,
$                    IERR )
      END IF
*
*      Return optimal workspace in WORK(1)
*
      WORK( 1 ) = DBLE( MAXWRK )
*
      RETURN
*
*      End of DGESDD
*
      END

```

— LAPACK dgesdd —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgesdd (jobz m n a lda s u ldu vt ldvt work lwork iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
              (type (simple-array double-float (*)) work vt u s a)
              (type fixnum info lwork ldvt ldu lda n m)
              (type character jobz))
    (f2cl-lib:with-multi-array-data
      ((jobz character jobz-%data% jobz-%offset%)
       (a double-float a-%data% a-%offset%)
       (s double-float s-%data% s-%offset%)
       (u double-float u-%data% u-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (work double-float work-%data% work-%offset%)
       (iwork fixnum iwork-%data% iwork-%offset%)))
    (prog ((dum (make-array 1 :element-type 'double-float))
           (idum (make-array 1 :element-type 'fixnum)) (anrm 0.0)
           (bignum 0.0) (eps 0.0) (smlnum 0.0) (bdspac 0) (blk 0) (chunk 0)
           (i 0) (ie 0) (ierr 0) (il 0) (ir 0) (iscl 0) (itau 0) (itaup 0)
           (itauq 0) (iu 0) (ivt 0) (ldwkvvt 0) (ldwrkl 0) (ldwrkr 0)
           (ldwrku 0) (maxwrk 0) (minmn 0) (minwrk 0) (mnthr 0) (nwork 0)
           (wrkbl 0) (lquery nil) (wntqa nil) (wntqas nil) (wntqn nil)
           (wntqo nil) (wntqs nil)))

```

```

(declare (type (simple-array double-float (1)) dum)
  (type (simple-array fixnum (1)) idum)
  (type (double-float) anrm bignum eps smlnum)
  (type fixnum bdspac blk chunk i ie ierr il ir
    iscl itau itaup itauq iu ivt ldwkvtr
    ldwrkl ldwrkr ldwrku maxwrk minmn
    minwrk mnthr nwork wrkbl)
  (type (member t nil) lquery wntqa wntqas wntqn wntqo wntqs))

(setf info 0)
(setf minmn (min (the fixnum m) (the fixnum n)))
(setf mnthr (f2cl-lib:int (/ (* minmn 11.0) 6.0)))
(setf wntqa (char-equal jobz #\A))
(setf wntqs (char-equal jobz #\S))
(setf wntqas (or wntqa wntqs))
(setf wntqo (char-equal jobz #\O))
(setf wntqn (char-equal jobz #\N))
(setf minwrk 1)
(setf maxwrk 1)
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((not (or wntqa wntqs wntqo wntqn))
    (setf info -1))
  ((< m 0)
    (setf info -2))
  ((< n 0)
    (setf info -3))
  ((< lda (max (the fixnum 1) (the fixnum m)))
    (setf info -5))
  ((or (< ldu 1) (and wntqas (< ldu m)) (and wntqo (< m n) (< ldu m)))
    (setf info -8))
  ((or (< ldvt 1)
    (and wntqa (< ldvt n))
    (and wntqs (< ldvt minmn))
    (and wntqo (>= m n) (< ldvt n)))
    (setf info -10)))
(cond
  ((and (= info 0) (> m 0) (> n 0))
    (cond
      ((>= m n)
        (cond
          (wntqn
            (setf bdspac (f2cl-lib:int-mul 7 n)))
          (t
            (setf bdspac
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 n n)
                (f2cl-lib:int-mul 4 n))))))
      (t
        (cond
          ((>= m mnthr)
            (cond
              (wntqn
                (setf bdspac (f2cl-lib:int-mul 7 n))
                (setf minwrk (f2cl-lib:int-min minwrk bdspac))
                (setf maxwrk (f2cl-lib:int-max maxwrk bdspac))
                (setf mnthr mnthr))
              (t
                (setf bdspac (f2cl-lib:int-mul 7 n))
                (setf minwrk (f2cl-lib:int-min minwrk bdspac))
                (setf maxwrk (f2cl-lib:int-max maxwrk bdspac))
                (setf mnthr mnthr)))))
          (t
            (setf bdspac (f2cl-lib:int-mul 7 n))
            (setf minwrk (f2cl-lib:int-min minwrk bdspac))
            (setf maxwrk (f2cl-lib:int-max maxwrk bdspac))
            (setf mnthr mnthr)))))))
  (t
    (setf info -1)))

```

```

(setf wrkbl
  (f2cl-lib:int-add n
    (f2cl-lib:int-mul n
      (ilaenv 1
        "DGEQRF" " " " m
        n -1 -1))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
            (ilaenv
              1
              "DGEBRD"
              " "
              n n
              -1
              -1))))))

(setf maxwrk
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspac n))))
(setf minwrk (f2cl-lib:int-add bdspac n))
(wntqo
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
          n -1 -1))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGQR"
            " "
            m n
            n
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
            (ilaenv

```

```

1
"DGEBRD"
" "
n n
-1
-1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "QLN"
            n n
            n
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            n
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspace
        (f2cl-lib:int-mul 3
          n))))))

(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul 2 n n)))
(setf minwrk
  (f2cl-lib:int-add bdspace
    (f2cl-lib:int-mul 2 n n)
    (f2cl-lib:int-mul 3 n))))

(wntqs
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
            n -1 -1))))

(setf wrkbl

```



```

(max (the fixnum wrkbl)
  (the fixnum
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv
          1
          "DORGQR"
          " "
          m n
          n
          -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
          (ilaenv
            1
            "DGEBRD"
            " "
            n n
            -1
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "QLN"
            n n
            n
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            n
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            n
            -1))))))

```

```

        (the fixnum
          (f2cl-lib:int-add bdspace
            (f2cl-lib:int-mul 3
              n))))))
(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul n n)))
(setf minwrk
  (f2cl-lib:int-add bdspace
    (f2cl-lib:int-mul n n)
    (f2cl-lib:int-mul 3 n))))
(wntqa
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
            n -1 -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul m
          (ilaenv 1
            "DORGQR" " " " m
              m m
                n
                  -1)))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
            (ilaenv 1
              "DGEBRD"
              " "
                n n
                  -1
                    -1)))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv 1
            "DORMBR"

```

```

                                "QLN"
                                n n
                                n
                                -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            n
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspace
        (f2cl-lib:int-mul 3
          n))))))

(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul n n))
(setf minwrk
  (f2cl-lib:int-add bdspace
    (f2cl-lib:int-mul n n)
    (f2cl-lib:int-mul 3 n))))))

(t
  (setf wrkbl
    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
      (f2cl-lib:int-mul
        (f2cl-lib:int-add m n)
        (ilaenv 1 "DGEHRD" " " " m n -1 -1))))

(cond
  (wntqn
    (setf maxwrk
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspace
            (f2cl-lib:int-mul 3
              n))))))

    (setf minwrk
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (max (the fixnum m)
          (the fixnum
            bdspace))))))

  (wntqo
    (setf wrkbl
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (max (the fixnum m)
              (the fixnum
                bdspace))))))

```

```

        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                            (f2cl-lib:int-mul n
                              (ilaenv
                                1
                                "DORMBR"
                                "QLN"
                                m n
                                n
                                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                            (f2cl-lib:int-mul n
                              (ilaenv
                                1
                                "DORMBR"
                                "PRT"
                                n n
                                n
                                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspace
                            (f2cl-lib:int-mul 3
                                                  n))))))
(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m n)))
(setf minwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                    (max (the fixnum m)
                        (the fixnum
                          (f2cl-lib:int-add
                            (f2cl-lib:int-mul n n)
                            bdspace))))))
(wntqs
  (setf wrkbl
    (max (the fixnum wrkbl)
          (the fixnum
            (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                              (f2cl-lib:int-mul n
                                (ilaenv
                                  1
                                  "DORMBR"
                                  "QLN"
                                  m n
                                  n
                                  -1))))))

```

```

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            n
            -1))))))

(setf maxwrk
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspace
        (f2cl-lib:int-mul 3
          n))))))

(setf minwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
    (max (the fixnum m)
      (the fixnum
        bdspace))))))

(wntqa
  (setf wrkbl
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (f2cl-lib:int-mul m
            (ilaenv
              1
              "DORMBR"
              "QLN"
              m m
              n
              -1))))))

  (setf wrkbl
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (f2cl-lib:int-mul n
            (ilaenv
              1
              "DORMBR"
              "PRT"
              n n
              n
              -1))))))

  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (f2cl-lib:int-mul n
            (ilaenv
              1
              "DORMBR"
              "PRT"
              n n
              n
              -1))))))

```

```

        (the fixnum
          (f2cl-lib:int-add bdspac
            (f2cl-lib:int-mul 3
              n))))))
      (setf minwrk
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (max (the fixnum m)
            (the fixnum
              bdspac)))))))))
(t
  (cond
    (wntqn
      (setf bdspac (f2cl-lib:int-mul 7 m)))
    (t
      (setf bdspac
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m m)
          (f2cl-lib:int-mul 4 m))))))
(cond
  ((>= n mnthr)
    (cond
      (wntqn
        (setf wrkbl
          (f2cl-lib:int-add m
            (f2cl-lib:int-mul m
              (ilaenv 1
                "DGELQF" " " " m
                  n -1 -1))))))
      (setf wrkbl
        (max (the fixnum wrkbl)
          (the fixnum
            (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
              (f2cl-lib:int-mul 2
                m
                  (ilaenv 1
                    "DGEBRD"
                    " " "
                      m m
                        -1
                          -1)))))))
    (setf maxwrk
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspac m))))
    (setf minwrk (f2cl-lib:int-add bdspac m)))
  (wntqo
    (setf wrkbl
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul m
          (ilaenv 1

```

```

                                "DGELQF" " " " m
                                n -1 -1))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGLQ"
            " "
            m n
            m
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1
            "DGEBRD"
            " "
            m m
            -1
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "QLN"
            m m
            m
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "PRT"
            m m
            m

```

```
(-1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspac
                             (f2cl-lib:int-mul 3
                                                  m))))))
(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul 2 m m)))
(setf minwrk
  (f2cl-lib:int-add bdspac
                    (f2cl-lib:int-mul 2 m m)
                    (f2cl-lib:int-mul 3 m))))
(wntqs
 (setf wrkbl
   (f2cl-lib:int-add m
                     (f2cl-lib:int-mul m
                                         (ilaenv 1
                                                  "DGELQF" " " m
                                                  n -1 -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add m
                             (f2cl-lib:int-mul m
                                                  (ilaenv 1
                                                           1
                                                           "DORGLQ"
                                                           " "
                                                           m n
                                                           m
                                                           -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                             (f2cl-lib:int-mul 2
                                                  m
                                                  (ilaenv 1
                                                           1
                                                           "DGEHRD"
                                                           " "
                                                           m m
                                                           -1
                                                           -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                             (f2cl-lib:int-mul m
```



```

(ilaenv
  1
  "DORMBR"
  "QLN"
  m m
  m
  -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "PRT"
            m m
            m
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspac
        (f2cl-lib:int-mul 3
          m))))))

(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m m)))
(setf minwrk
  (f2cl-lib:int-add bdspac
    (f2cl-lib:int-mul m m)
    (f2cl-lib:int-mul 3 m))))

(wntqa
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " " m
          n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGLQ"
            " "
            n n
            m
            -1)))))))

```

```

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1
            "DGEBRD"
            " "
            m m
            -1
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "QLN"
            m m
            m
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "PRT"
            m m
            m
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspace
        (f2cl-lib:int-mul 3
          m))))))

(setf maxwrk
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m m)))
(setf minwrk
  (f2cl-lib:int-add bdspace
    (f2cl-lib:int-mul m m)
    (f2cl-lib:int-mul 3 m))))))

(t

```

```

(setf wrkbl
  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
    (f2cl-lib:int-mul
      (f2cl-lib:int-add m n)
      (ilaenv 1 "DGEBRD" " " m n -1 -1))))
(cond
  (wntqn
    (setf maxwrk
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspac
            (f2cl-lib:int-mul 3
              m))))))
    (setf minwrk
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (max (the fixnum n)
          (the fixnum
            bdspac))))))
  (wntqo
    (setf wrkbl
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
            (f2cl-lib:int-mul m
              (ilaenv
                1
                "DORMBR"
                "QLN"
                m m
                n
                -1))))))
    (setf wrkbl
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
            (f2cl-lib:int-mul m
              (ilaenv
                1
                "DORMBR"
                "PRT"
                m n
                m
                -1))))))
    (setf wrkbl
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add bdspac
            (f2cl-lib:int-mul 3
              m))))))
    (setf maxwrk

```

```

(f2cl-lib:int-add wrkbl (f2cl-lib:int-mul m n)))
(setf minwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
    (max (the fixnum n)
      (the fixnum
        (f2cl-lib:int-add
          (f2cl-lib:int-mul m m)
          bdspac))))))
(wntqs
  (setf wrkbl
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
          (f2cl-lib:int-mul m
            (ilaenv
              1
              "DORMBR"
              "QLN"
              m m
              n
              -1)))))))
  (setf wrkbl
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
          (f2cl-lib:int-mul m
            (ilaenv
              1
              "DORMBR"
              "PRT"
              m n
              m
              -1)))))))
  (setf maxwrk
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add bdspac
          (f2cl-lib:int-mul 3
            m))))))
  (setf minwrk
    (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
      (max (the fixnum n)
        (the fixnum
          bdspac))))))
(wntqa
  (setf wrkbl
    (max (the fixnum wrkbl)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
          (f2cl-lib:int-mul m

```

```

(ilaenv
  1
  "DORMBR"
  "QLN"
  m m
  n
  -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORMBR"
            "PRT"
            n n
            m
            -1)))))))

(setf maxwrk
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add bdspac
        (f2cl-lib:int-mul 3
          m))))))

(setf minwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
    (max (the fixnum n)
      (the fixnum
        bdspac))))))

(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (the fixnum maxwrk) 'double-float)))

(cond
  ((and (< lwork minwrk) (not lquery))
    (setf info -12)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGESDD" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(cond
  ((or (= m 0) (= n 0))
    (if (>= lwork 1)
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one))
    (go end_label)))
(setf eps (dlamch "P"))
(setf smlnum (/ (f2cl-lib:fsqrt (dlamch "S")) eps))

```

```

(setf bignum (/ one smlnum))
(setf anrm (dlang "M" m n a lda dum))
(setf iscl 0)
(cond
  ((and (> anrm zero) (< anrm smlnum))
    (setf iscl 1)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 anrm smlnum m n a lda ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8))
      (setf ierr var-9)))
    (> anrm bignum)
    (setf iscl 1)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 anrm bignum m n a lda ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8))
      (setf ierr var-9))))
(cond
  ((>= m n)
    (cond
      ((>= m mnthr)
        (cond
          (wntqn
            (setf itau 1)
            (setf nwork (f2cl-lib:int-add itau n))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
              (dgeqrf m n a lda
                (f2cl-lib:array-slice work double-float (itau) ((1 *)))
                (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
              (setf ierr var-7))
            (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
              zero
              (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
              lda)
            (setf ie 1)
            (setf itauq (f2cl-lib:int-add ie n))
            (setf itaup (f2cl-lib:int-add itauq n))
            (setf nwork (f2cl-lib:int-add itaup n))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
                var-9 var-10)
              (dgebrd n n a lda s
                (f2cl-lib:array-slice work double-float (ie) ((1 *)))
                (f2cl-lib:array-slice work double-float (itauq) ((1 *)))

```

```

(f2cl-lib:array-slice work double-float (itau) ((1 *)))
(f2cl-lib:array-slice work double-float (nwork) ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
               var-7 var-8 var-9))
(setf ierr var-10))
(setf nwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "N" n s
   (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
   1 dum 1 dum idum
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13)))
(wntqo
 (setf ir 1)
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul lda n)
                      (f2cl-lib:int-mul n n)
                      (f2cl-lib:int-mul 3 n)
                      bdspace))
   (setf ldwrkr lda))
  (t
   (setf ldwrkr
    (the fixnum
     (truncate (- lwork (* n n) (* 3 n) bdspace)
               n)))))
 (setf itau (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
 (setf nwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
   (f2cl-lib:array-slice work double-float (itau) ((1 *)))
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
 (dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
  zero
  (f2cl-lib:array-slice work double-float ((+ ir 1)) ((1 *)))
  ldwrkr)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)

```

```

(dorgqr m n n a lda
  (f2cl-lib:array-slice work double-float (itau) ((1 *)))
  (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
  var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf nwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10)
  (dgebrd n n
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9))
  (setf ierr var-10))
(setf iu nwork)
(setf nwork (f2cl-lib:int-add iu (f2cl-lib:int-mul n n)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" n s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
    vt ldvt dum idum
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" n n n
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12 var-13))
  (setf ierr var-13))

```



```

                                var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" n n n
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrkr))
  (> i m) nil)
(tagbody
  (setf chunk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
      (the fixnum ldwrkr)))
  (dgemm "N" "N" chunk n n one
    (f2cl-lib:array-slice a
      double-float
      (i 1)
      ((1 lda) (1 *)))
    lda (f2cl-lib:array-slice work double-float (iu) ((1 *)))
    n zero
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr)
  (dlacpy "F" chunk n
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice a
      double-float
      (i 1)
      ((1 lda) (1 *)))
    lda)))
(wntqs
  (setf ir 1)
  (setf ldwrkr n)
  (setf itau (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
  (setf nwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))

```

```

        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
      (setf ierr var-7))
    (dlacpy "U" n n a lda
      (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
    (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
      zero
      (f2cl-lib:array-slice work double-float ((+ ir 1)) ((1 *)))
      ldwrkr)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (dorgqr m n n a lda
        (f2cl-lib:array-slice work double-float (itau) ((1 *)))
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7))
      (setf ierr var-8))
    (setf ie itau)
    (setf itauq (f2cl-lib:int-add ie n))
    (setf itaup (f2cl-lib:int-add itauq n))
    (setf nwork (f2cl-lib:int-add itaup n))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10)
      (dgebrd n n
        (f2cl-lib:array-slice work double-float (ir) ((1 *)))
        ldwrkr s
        (f2cl-lib:array-slice work double-float (ie) ((1 *)))
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7 var-8 var-9))
      (setf ierr var-10))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)
      (dbdsdc "U" "I" n s
        (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
        ldu vt ldvt dum idum
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        iwork info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7 var-8 var-9 var-10 var-11 var-12))
      (setf info var-13))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11 var-12 var-13)

```

```

(dormbr "Q" "L" "N" n n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr
  (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
  u ldu
  (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
  var-7 var-8 var-9 var-10 var-11 var-12))
(setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" n n n
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(dlacpy "F" n n u ldu
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr)
(dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *))) ldwrkr
  zero u ldu))
(wntqa
  (setf iu 1)
  (setf ldwrku n)
  (setf itau (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
  (setf nwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
    (setf ierr var-7))
  (dlacpy "L" m n a lda u ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dorgqr m m n u ldu
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7))
    (setf ierr var-8))
  (setf ierr var-9))

```

```

(setf ierr var-8))
(dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
zero
(f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
lda)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf nwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10)
  (dgebrd n n a lda s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
   (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" n s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (iu) ((1 *))) n
   vt ldvt dum idum
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" n n n a lda
   (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
   (f2cl-lib:array-slice work double-float (iu) ((1 *)))
   ldwrku
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" n n n a lda
   (f2cl-lib:array-slice work double-float (itaup) ((1 *)))

```

```

      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))
  (dgemm "N" "N" m n n one u ldu
    (f2cl-lib:array-slice work double-float (iu) ((1 *))) ldwrku
    zero a lda)
  (dlacpy "F" m n a lda u ldu)))
(t
  (setf ie 1)
  (setf itauq (f2cl-lib:int-add ie n))
  (setf itaup (f2cl-lib:int-add itauq n))
  (setf nwork (f2cl-lib:int-add itaup n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10)
    (dgebrd m n a lda s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9))
    (setf ierr var-10))
  (cond
    (wntqn
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9 var-10 var-11 var-12 var-13)
        (dbdsdc "U" "N" n s
          (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
          1 dum 1 dum idum
          (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
          iwork info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
          var-7 var-8 var-9 var-10 var-11 var-12))
        (setf info var-13)))
      (wntqo
        (setf iu nwork)
        (cond
          ((>= lwork
            (f2cl-lib:int-add (f2cl-lib:int-mul m n)
              (f2cl-lib:int-mul 3 n)
              bdspace))
            (setf ldwrku m)
            (setf nwork
              (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))

```

```

(dlaset "F" m n zero zero
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku))
(t
 (setf ldwrku n)
 (setf nwork
  (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
 (setf ir nwork)
 (setf ldwrkr
  (the fixnum
   (truncate (- lwork (* n n) (* 3 n) n))))
 (setf nwork (f2cl-lib:int-add iu (f2cl-lib:int-mul ldwrku n)))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" n s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (iu) ((1 *)))
   ldwrku vt ldvt dum idum
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" n n n a lda
   (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
   vt ldvt
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                      (f2cl-lib:int-mul 3 n)
                      bdspac))
   (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m n n a lda
     (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
     (f2cl-lib:array-slice work double-float (iu) ((1 *)))
     ldwrku

```

```

(f2cl-lib:array-slice work
  double-float
  (nwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12))
(setf ierr var-13))
(dlacpy "F" m n
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku a lda))
(t
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
(dorgbr "Q" m n n a lda
(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (nwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrkr))
  (> i m) nil)
(tagbody
  (setf chunk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub m i)
          1))
      (the fixnum ldwrkr)))
(dgemm "N" "N" chunk n n one
(f2cl-lib:array-slice a
  double-float
  (i 1)
  ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku zero
(f2cl-lib:array-slice work double-float (ir) ((1 *)))
ldwrkr)

```

```

(dlacpy "F" chunk n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr
  (f2cl-lib:array-slice a
    double-float
    (i 1)
    ((1 lda) (1 *)))
  lda))))))
(wntqs
  (dlaset "F" m n zero zero u ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" n s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m n n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" n n n a lda
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13)))
(wntqa
  (dlaset "F" m m zero zero u ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "U" "I" n s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u

```



```

        ldu vt ldvt dum idum
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
  (dlaset "F" (f2cl-lib:int-sub m n) (f2cl-lib:int-sub m n) zero
one
(f2cl-lib:array-slice u
                      double-float
                      ((+ n 1) (f2cl-lib:int-add n 1))
                      ((1 ldu) (1 *)))

  ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" n n m a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      vt ldvt
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))))))

  (t
    (cond
      ((>= n mnthr)
        (cond
          (wntqn
            (setf itau 1)
            (setf nwork (f2cl-lib:int-add itau m))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
              (dgelqf m n a lda
                (f2cl-lib:array-slice work double-float (itau) ((1 *)))
                (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
                (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
              (setf ierr var-7))

```

```

(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
zero
(f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
lda)
(setf ie 1)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10)
  (dgebrd m m a lda s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
   (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9))
  (setf ierr var-10))
(setf nwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "N" m s
   (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
   1 dum 1 dum idum
   (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
   iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13)))
(wntqo
  (setf ivt 1)
  (setf il (f2cl-lib:int-add ivt (f2cl-lib:int-mul m m)))
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul m n)
        (f2cl-lib:int-mul m m)
        (f2cl-lib:int-mul 3 m)
        bdspace))
      (setf ldwrkl m)
      (setf chunk n))
    (t
      (setf ldwrkl m)
      (setf chunk
        (the fixnum
          (truncate (- lwork (* m m)) m))))))
  (setf itau (f2cl-lib:int-add il (f2cl-lib:int-mul ldwrkl m)))
  (setf nwork (f2cl-lib:int-add itau m))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
  (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
  (f2cl-lib:array-slice work
    double-float
    ((+ il ldwrkl))
    ((1 *)))
  ldwrkl)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (dorglq m n m a lda
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10)
  (dgebrd m m
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" m s
    (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
    ldu

```

```

(f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
dum idum
(f2cl-lib:array-slice work double-float (nwork) ((1 *)))
iwork info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9 var-10 var-11 var-12))
(setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" m m m
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    u ldu
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" m m m
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    (f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
  (> i n) nil)
(tagbody
  (setf blk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
      (the fixnum chunk)))
  (dgemm "N" "N" m blk m one
    (f2cl-lib:array-slice work double-float (ivt) ((1 *))) m
    (f2cl-lib:array-slice a
      double-float
      (1 i)
      ((1 lda) (1 *)))
    lda zero
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl)

```

```

(dlacpy "F" m blk
  (f2cl-lib:array-slice work double-float (il) ((1 *)))
  ldwrkl
  (f2cl-lib:array-slice a
    double-float
    (1 i)
    ((1 lda) (1 *)))
  lda)))
(wntqs
  (setf il 1)
  (setf ldwrkl m)
  (setf itau (f2cl-lib:int-add il (f2cl-lib:int-mul ldwrkl m)))
  (setf nwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
    (setf ierr var-7))
  (dlacpy "L" m m a lda
    (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
  (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
    zero
    (f2cl-lib:array-slice work
      double-float
      ((+ il ldwrkl))
      ((1 *)))
    ldwrkl)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dorglq m n m a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7))
    (setf ierr var-8))
  (setf ie itau)
  (setf itauq (f2cl-lib:int-add ie m))
  (setf itaup (f2cl-lib:int-add itauq m))
  (setf nwork (f2cl-lib:int-add itaup m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10)
    (dgebrd m m
      (f2cl-lib:array-slice work double-float (il) ((1 *)))
      ldwrkl s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))

```

```

(f2cl-lib:array-slice work double-float (itauq) ((1 *)))
(f2cl-lib:array-slice work double-float (itaup) ((1 *)))
(f2cl-lib:array-slice work double-float (nwork) ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
               var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" m s
    (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
    ldu vt ldvt dum idum
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    iwork info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" m m m
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    u ldu
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" m m m
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    ldwrkl
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(dlacpy "F" m m vt ldvt
  (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl)
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (il) ((1 *))) ldwrkl
  a lda zero vt ldvt))
(wntqa

```

```

(setf ivt 1)
(setf ldwkv t m)
(setf itau (f2cl-lib:int-add ivt (f2cl-lib:int-mul ldwkv t m)))
(setf nwork (f2cl-lib:int-add itau m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (dorglq n n m vt ldvt
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7))
  (setf ierr var-8))
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
  zero
  (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
  lda)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf nwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10)
  (dgebrd m m a lda s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "U" "I" m s
    (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
    ldu
    (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
    ldwkv t dum idum

```

```

(f2cl-lib:array-slice work double-float (nwork) ((1 *)))
iwork info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9 var-10 var-11 var-12))
(setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" m m m a lda
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
    u ldu
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" m m m a lda
    (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
    (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
    ldwkv
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (ivt) ((1 *))) ldwkv
  vt ldvt zero a lda)
(dlacpy "F" m n a lda vt ldvt))))
(t
  (setf ie 1)
  (setf itauq (f2cl-lib:int-add ie m))
  (setf itaup (f2cl-lib:int-add itauq m))
  (setf nwork (f2cl-lib:int-add itaup m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10)
    (dgebrd m n a lda s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9))
    (setf ierr var-10))
  (cond

```



```

(wntqn
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "L" "N" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
      1 dum 1 dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13)))
(wntqo
  (setf ldwkv t m)
  (setf iwork nwork)
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                        (f2cl-lib:int-mul 3 m)
                        bdspace))
     (dlaset "F" m n zero zero
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      ldwkv t)
     (setf nwork
      (f2cl-lib:int-add iwork (f2cl-lib:int-mul ldwkv t n))))
    (t
     (setf nwork
      (f2cl-lib:int-add iwork (f2cl-lib:int-mul ldwkv t m)))
     (setf il nwork)
     (setf chunk
      (the fixnum
       (truncate (- lwork (* m m) (* 3 m)) m))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dbdsdc "L" "I" m s
    (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
    ldu
    (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
    ldwkv t dum idum
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                     var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "L" "N" m m n a lda
    (f2cl-lib:array-slice work double-float (itauq) ((1 *)))

```

```

u ldu
  (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
              var-7 var-8 var-9 var-10 var-11 var-12))
(setf ierr var-13))
(cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul m n)
                      (f2cl-lib:int-mul 3 m)
                      bdspace))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13)
      (dormbr "P" "R" "T" m n m a lda
        (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
        (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
        ldwkv
        (f2cl-lib:array-slice work
                              double-float
                              (nwork)
                              ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8 var-9 var-10 var-11
                      var-12))
      (setf ierr var-13))
    (dlacpy "F" m n
      (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
      ldwkv a lda))
  (t
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
      (dorgbr "P" m n m a lda
        (f2cl-lib:array-slice work
                              double-float
                              (itaup)
                              ((1 *)))
        (f2cl-lib:array-slice work
                              double-float
                              (nwork)
                              ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1)
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5

```

```

                                var-6 var-7 var-8))
  (setf ierr var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
  (> i n) nil)
(tagbody
  (setf blk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub n i)
          1))
      (the fixnum chunk)))
  (dgemm "N" "N" m blk m one
    (f2cl-lib:array-slice work double-float (ivt) ((1 *)))
    ldwkv
    (f2cl-lib:array-slice a
      double-float
      (1 i)
      ((1 lda) (1 *)))

    lda zero
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    m)
  (dlacpy "F" m blk
    (f2cl-lib:array-slice work double-float (il) ((1 *)))
    m
    (f2cl-lib:array-slice a
      double-float
      (1 i)
      ((1 lda) (1 *)))

    lda))))))
(wntqs
  (dlaset "F" m n zero zero vt ldvt)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "L" "I" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9 var-10 var-11 var-12))
(setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "R" "T" m n m a lda
    (f2cl-lib:array-slice work double-float (itau) ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12))
  (setf ierr var-13)))
(wntqa
  (dlaset "F" n n zero zero vt ldvt)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dbdsdc "L" "I" m s
      (f2cl-lib:array-slice work double-float (ie) ((1 *))) u
      ldu vt ldvt dum idum
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      iwork info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
    (setf info var-13))
  (dlaset "F" (f2cl-lib:int-sub n m) (f2cl-lib:int-sub n m) zero
    one
    (f2cl-lib:array-slice vt
      double-float
      ((+ m 1) (f2cl-lib:int-add m 1))
      ((1 ldvt) (1 *)))
  ldvt)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "Q" "L" "N" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      u ldu
      (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12))
    (setf ierr var-13))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13)
    (dormbr "P" "R" "T" n n m a lda
      (f2cl-lib:array-slice work double-float (itau) ((1 *)))

```

```

        vt ldvt
        (f2cl-lib:array-slice work double-float (nwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork nwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
              var-7 var-8 var-9 var-10 var-11 var-12))
(setf ierr var-13))))))
(cond
  ((= iscl 1)
   (if (> anrm bignum)
       (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9)
         (dlascl "G" 0 0 bignum anrm minmn 1 s minmn ierr)
         (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                          var-7 var-8))
         (setf ierr var-9)))
       (if (< anrm smlnum)
           (multiple-value-bind
             (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
              var-9)
             (dlascl "G" 0 0 smlnum anrm minmn 1 s minmn ierr)
             (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                              var-7 var-8))
             (setf ierr var-9))))
       (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (coerce (realpart maxwrk) 'double-float)))
  end_label
  (return
   (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dgesvd LAPACK

— dgesvd.input —

```

)set break resume
)sys rm -f dgesvd.output
)spool dgesvd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgesvd.help —

=====

dgesvd examples

=====

=====

Man Page Details

=====

NAME

DGESVD - the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors

SYNOPSIS

```
SUBROUTINE DGESVD( JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT,
                   WORK, LWORK, INFO )
```

CHARACTER JOBU, JOBVT

INTEGER INFO, LDA, LDU, LDVT, LWORK, M, N

DOUBLE PRECISION A(LDA, *), S(*), U(LDU, *), VT(LDVT, *), WORK(*)

Purpose

=====

DGESVD computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

$$A = U * \text{SIGMA} * \text{transpose}(V)$$

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal matrix, and V is an N-by-N orthogonal matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V**T, not V.

Arguments

=====

JOBU (input) CHARACTER*1

Specifies options for computing all or part of the matrix U:
 = 'A': all M columns of U are returned in array U;
 = 'S': the first min(m,n) columns of U (the left singular vectors) are returned in the array U;
 = 'O': the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A;
 = 'N': no columns of U (no left singular vectors) are computed.

JOBVT (input) CHARACTER*1
 Specifies options for computing all or part of the matrix V**T:
 = 'A': all N rows of V**T are returned in the array VT;
 = 'S': the first min(m,n) rows of V**T (the right singular vectors) are returned in the array VT;
 = 'O': the first min(m,n) rows of V**T (the right singular vectors) are overwritten on the array A;
 = 'N': no rows of V**T (no right singular vectors) are computed.

JOBVT and JOBU cannot both be 'O'.

M (input) INTEGER
 The number of rows of the input matrix A. $M \geq 0$.

N (input) INTEGER
 The number of columns of the input matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the M-by-N matrix A.
 On exit,
 if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise);
 if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V**T (the right singular vectors, stored rowwise);
 if JOBU .ne. 'O' and JOBVT .ne. 'O', the contents of A are destroyed.

LDA (input) INTEGER
 The leading dimension of the array A. $LDA \geq \max(1,M)$.

S (output) DOUBLE PRECISION array, dimension (min(M,N))
 The singular values of A, sorted so that $S(i) \geq S(i+1)$.

U (output) DOUBLE PRECISION array, dimension (LDU,UCOL)
 (LDU,M) if JOBU = 'A' or (LDU,min(M,N)) if JOBU = 'S'.
 If JOBU = 'A', U contains the M-by-M orthogonal matrix U;
 if JOBU = 'S', U contains the first min(m,n) columns of U

(the left singular vectors, stored columnwise);
 if JOBU = 'N' or 'O', U is not referenced.

- LDU (input) INTEGER
 The leading dimension of the array U. LDU ≥ 1 ; if
 JOBU = 'S' or 'A', LDU $\geq M$.
- VT (output) DOUBLE PRECISION array, dimension (LDVT,N)
 If JOBVT = 'A', VT contains the N-by-N orthogonal matrix
 $V^{*}T$;
 if JOBVT = 'S', VT contains the first $\min(m,n)$ rows of
 $V^{*}T$ (the right singular vectors, stored rowwise);
 if JOBVT = 'N' or 'O', VT is not referenced.
- LDVT (input) INTEGER
 The leading dimension of the array VT. LDVT ≥ 1 ; if
 JOBVT = 'A', LDVT $\geq N$; if JOBVT = 'S', LDVT $\geq \min(M,N)$.
- WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
 On exit, if INFO = 0, WORK(1) returns the optimal LWORK;
 if INFO > 0 , WORK(2:MIN(M,N)) contains the unconverged
 superdiagonal elements of an upper bidiagonal matrix B
 whose diagonal is in S (not necessarily sorted). B
 satisfies $A = U * B * VT$, so it has the same singular values
 as A, and singular vectors related by U and VT.
- LWORK (input) INTEGER
 The dimension of the array WORK. LWORK ≥ 1 .
 LWORK $\geq \text{MAX}(3 * \text{MIN}(M,N) + \text{MAX}(M,N), 5 * \text{MIN}(M,N))$.
 For good performance, LWORK should generally be larger.
- If LWORK = -1, then a workspace query is assumed; the routine
 only calculates the optimal size of the WORK array, returns
 this value as the first entry of the WORK array, and no error
 message related to LWORK is issued by XERBLA.
- INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: if DBDSQR did not converge, INFO specifies how many
 superdiagonals of an intermediate bidiagonal form B
 did not converge to zero. See the description of WORK
 above for details.


```

      SUBROUTINE DGESVD( JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT,
$                        WORK, LWORK, INFO )
*
*  -- LAPACK driver routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
*  CHARACTER          JOBU, JOBVT
*  INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION   A( LDA, * ), S( * ), U( LDU, * ),
$                    VT( LDVT, * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
*  DOUBLE PRECISION   ZERO, ONE
*  PARAMETER          ( ZERO = 0.0D0, ONE = 1.0D0 )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL            LQUERY, WNTUA, WNTUAS, WNTUN, WNTUO, WNTUS,
$                    WNTVA, WNTVAS, WNTVN, WNTVO, WNTVS
*  INTEGER            BDSPAC, BLK, CHUNK, I, IE, IERR, IR, ISCL,
$                    ITAU, ITAUP, ITAUQ, IU, IWORK, LDWRKR, LDWRKU,
$                    MAXWRK, MINMN, MINWRK, MNTHR, NCU, NCVT, NRU,
$                    NRV, WRKBL
*  DOUBLE PRECISION   ANRM, BIGNUM, EPS, SMLNUM
*
*  ..
*  .. Local Arrays ..
*  DOUBLE PRECISION   DUM( 1 )
*
*  ..
*  .. External Subroutines ..
*  EXTERNAL           DBDSQR, DGEBRD, DGELQF, DGEMM, DGEQRF, DLACPY,
$                    DLASCL, DLASET, DORGBR, DORGLQ, DORGQR, DORMBR,
$                    XERBLA
*
*  ..
*  .. External Functions ..
*  LOGICAL            LSAME
*  INTEGER            ILAENV
*  DOUBLE PRECISION   DLAMCH, DLANGE
*  EXTERNAL           LSAME, ILAENV, DLAMCH, DLANGE
*
*  ..
*  .. Intrinsic Functions ..
*  INTRINSIC          MAX, MIN, SQRT
*
*  ..
*  .. Executable Statements ..

```

```

*
*   Test the input arguments
*
      INFO = 0
      MINMN = MIN( M, N )
      MNTHR = ILAENV( 6, 'DGESVD', JOBU // JOBVT, M, N, 0, 0 )
      WNTUA = LSAME( JOBU, 'A' )
      WNTUS = LSAME( JOBU, 'S' )
      WNTUAS = WNTUA .OR. WNTUS
      WNTUO = LSAME( JOBU, 'O' )
      WNTUN = LSAME( JOBU, 'N' )
      WNTVA = LSAME( JOBVT, 'A' )
      WNTVS = LSAME( JOBVT, 'S' )
      WNTVAS = WNTVA .OR. WNTVS
      WNTVO = LSAME( JOBVT, 'O' )
      WNTVN = LSAME( JOBVT, 'N' )
      MINWRK = 1
      LQUERY = ( LWORK.EQ.-1 )

*
      IF( .NOT.( WNTUA .OR. WNTUS .OR. WNTUO .OR. WNTUN ) ) THEN
          INFO = -1
      ELSE IF( .NOT.( WNTVA .OR. WNTVS .OR. WNTVO .OR. WNTVN ) .OR.
$          ( WNTVO .AND. WNTUO ) ) THEN
          INFO = -2
      ELSE IF( M.LT.0 ) THEN
          INFO = -3
      ELSE IF( N.LT.0 ) THEN
          INFO = -4
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
          INFO = -6
      ELSE IF( LDU.LT.1 .OR. ( WNTUAS .AND. LDU.LT.M ) ) THEN
          INFO = -9
      ELSE IF( LDVT.LT.1 .OR. ( WNTVA .AND. LDVT.LT.N ) .OR.
$          ( WNTVS .AND. LDVT.LT.MINMN ) ) THEN
          INFO = -11
      END IF

*
*   Compute workspace
*   (Note: Comments in the code beginning "Workspace:" describe the
*   minimal amount of workspace needed at that point in the code,
*   as well as the preferred amount for good performance.
*   NB refers to the optimal block size for the immediately
*   following subroutine, as returned by ILAENV.)
*
      IF( INFO.EQ.0 .AND. ( LWORK.GE.1 .OR. LQUERY ) .AND. M.GT.0 .AND.
$          N.GT.0 ) THEN
          IF( M.GE.N ) THEN

*
*           Compute space needed for DBDSQR
*

```

```

      BDSPAC = 5*N
      IF( M.GE.MNTHR ) THEN
        IF( WNTUN ) THEN
*
*          Path 1 (M much larger than N, JOBU='N')
*
          MAXWRK = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1,
$             -1 )
          MAXWRK = MAX( MAXWRK, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          IF( WNTVO .OR. WNTVAS )
$             MAXWRK = MAX( MAXWRK, 3*N+( N-1 )*
$             ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
          MAXWRK = MAX( MAXWRK, BDSPAC )
          MINWRK = MAX( 4*N, BDSPAC )
          MAXWRK = MAX( MAXWRK, MINWRK )
        ELSE IF( WNTUO .AND. WNTVN ) THEN
*
*          Path 2 (M much larger than N, JOBU='O', JOBVT='N')
*
          WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
          WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$             N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, BDSPAC )
          MAXWRK = MAX( N*N+WRKBL, N*N+M*N+N )
          MINWRK = MAX( 3*N+M, BDSPAC )
          MAXWRK = MAX( MAXWRK, MINWRK )
        ELSE IF( WNTUO .AND. WNTVAS ) THEN
*
*          Path 3 (M much larger than N, JOBU='O', JOBVT='S' or
*          'A')
*
          WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
          WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$             N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+2*N*
$             ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+N*
$             ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, 3*N+( N-1 )*
$             ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
          WRKBL = MAX( WRKBL, BDSPAC )
          MAXWRK = MAX( N*N+WRKBL, N*N+M*N+N )
          MINWRK = MAX( 3*N+M, BDSPAC )
          MAXWRK = MAX( MAXWRK, MINWRK )
        ELSE IF( WNTUS .AND. WNTVN ) THEN

```

```

*
*      Path 4 (M much larger than N, JOBU='S', JOBVT='N')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$      N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$      ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$      ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTUS .AND. WNTVO ) THEN
*
*      Path 5 (M much larger than N, JOBU='S', JOBVT='O')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$      N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$      ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$      ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+( N-1 )*
$      ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = 2*N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTUS .AND. WNTVAS ) THEN
*
*      Path 6 (M much larger than N, JOBU='S', JOBVT='S' or
*      'A')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+N*ILAENV( 1, 'DORGQR', ' ', M,
$      N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$      ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$      ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+( N-1 )*
$      ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTUA .AND. WNTVN ) THEN

```

```

*
*      Path 7 (M much larger than N, JOBU='A', JOBVT='N')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+M*ILAENV( 1, 'DORGQR', ' ', M,
$           M, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$           ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$           ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTUA .AND. WNTVO ) THEN

*
*      Path 8 (M much larger than N, JOBU='A', JOBVT='O')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+M*ILAENV( 1, 'DORGQR', ' ', M,
$           M, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$           ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$           ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+( N-1 )*
$           ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = 2*N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTUA .AND. WNTVAS ) THEN

*
*      Path 9 (M much larger than N, JOBU='A', JOBVT='S' or
*      'A')
*
      WRKBL = N + N*ILAENV( 1, 'DGEQRF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, N+M*ILAENV( 1, 'DORGQR', ' ', M,
$           M, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+2*N*
$           ILAENV( 1, 'DGEBRD', ' ', N, N, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+N*
$           ILAENV( 1, 'DORGBR', 'Q', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, 3*N+( N-1 )*
$           ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = N*N + WRKBL
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      END IF

```

```

ELSE
*
*      Path 10 (M at least N, but not much larger)
*
      MAXWRK = 3*N + ( M+N )*ILAENV( 1, 'DGEBRD', ' ', M, N,
$      -1, -1 )
      IF( WNTUS .OR. WNTUO )
$      MAXWRK = MAX( MAXWRK, 3*N+N*
$      ILAENV( 1, 'DORGBR', 'Q', M, N, N, -1 ) )
      IF( WNTUA )
$      MAXWRK = MAX( MAXWRK, 3*N+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, N, -1 ) )
      IF( .NOT.WNTVN )
$      MAXWRK = MAX( MAXWRK, 3*N+( N-1 )*
$      ILAENV( 1, 'DORGBR', 'P', N, N, N, -1 ) )
      MAXWRK = MAX( MAXWRK, BDSPAC )
      MINWRK = MAX( 3*N+M, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      END IF
ELSE
*
*      Compute space needed for DBDSQR
*
      BDSPAC = 5*M
      IF( N.GE.MNTHR ) THEN
        IF( WNTVN ) THEN
*
*      Path 1t(N much larger than M, JOBVT='N')
*
          MAXWRK = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1,
$          -1 )
          MAXWRK = MAX( MAXWRK, 3*M+2*M*
$          ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
          IF( WNTUO .OR. WNTUAS )
$          MAXWRK = MAX( MAXWRK, 3*M+M*
$          ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )
          MAXWRK = MAX( MAXWRK, BDSPAC )
          MINWRK = MAX( 4*M, BDSPAC )
          MAXWRK = MAX( MAXWRK, MINWRK )
          ELSE IF( WNTVO .AND. WNTUN ) THEN
*
*      Path 2t(N much larger than M, JOBU='N', JOBVT='O')
*
            WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
            WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$            N, M, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+2*M*
$            ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
            WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$            ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )

```

```

      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = MAX( M*M+WRKBL, M*M+M*N+M )
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVO .AND. WNTUAS ) THEN
*
*      Path 3t(N much larger than M, JOBU='S' or 'A',
*      JOBVT='O')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBCD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = MAX( M*M+WRKBL, M*M+M*N+M )
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVS .AND. WNTUN ) THEN
*
*      Path 4t(N much larger than M, JOBU='N', JOBVT='S')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBCD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVS .AND. WNTUO ) THEN
*
*      Path 5t(N much larger than M, JOBU='O', JOBVT='S')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBCD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )

```

```

      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = 2*M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVS .AND. WNTUAS ) THEN
*
*      Path 6t(N much larger than M, JOBU='S' or 'A',
*      JOBVT='S')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+M*ILAENV( 1, 'DORGLQ', ' ', M,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVA .AND. WNTUN ) THEN
*
*      Path 7t(N much larger than M, JOBU='N', JOBVT='A')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+N*ILAENV( 1, 'DORGLQ', ' ', N,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVA .AND. WNTUO ) THEN
*
*      Path 8t(N much larger than M, JOBU='O', JOBVT='A')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+N*ILAENV( 1, 'DORGLQ', ' ', N,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )

```



```

      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = 2*M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      ELSE IF( WNTVA .AND. WNTUAS ) THEN
*
*      Path 9t(N much larger than M, JOBU='S' or 'A',
*      JOBVT='A')
*
      WRKBL = M + M*ILAENV( 1, 'DGELQF', ' ', M, N, -1, -1 )
      WRKBL = MAX( WRKBL, M+N*ILAENV( 1, 'DORGLQ', ' ', N,
$      N, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+2*M*
$      ILAENV( 1, 'DGEBRD', ' ', M, M, -1, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'P', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )
      WRKBL = MAX( WRKBL, BDSPAC )
      MAXWRK = M*M + WRKBL
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      END IF
    ELSE
*
*      Path 10t(N greater than M, but not much larger)
*
      MAXWRK = 3*M + ( M+N )*ILAENV( 1, 'DGEBRD', ' ', M, N,
$      -1, -1 )
      IF( WNTVS .OR. WNTVO )
$      MAXWRK = MAX( MAXWRK, 3*M+M*
$      ILAENV( 1, 'DORGBR', 'P', M, N, M, -1 ) )
      IF( WNTVA )
$      MAXWRK = MAX( MAXWRK, 3*M+N*
$      ILAENV( 1, 'DORGBR', 'P', N, N, M, -1 ) )
      IF( .NOT.WNTUN )
$      MAXWRK = MAX( MAXWRK, 3*M+( M-1 )*
$      ILAENV( 1, 'DORGBR', 'Q', M, M, M, -1 ) )
      MAXWRK = MAX( MAXWRK, BDSPAC )
      MINWRK = MAX( 3*M+N, BDSPAC )
      MAXWRK = MAX( MAXWRK, MINWRK )
      END IF
      END IF
      WORK( 1 ) = MAXWRK
    END IF
*
    IF( LWORK.LT.MINWRK .AND. .NOT.LQUERY ) THEN
      INFO = -13
    END IF
    IF( INFO.NE.0 ) THEN

```

```

        CALL XERBLA( 'DGESVD', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Quick return if possible
*
    IF( M.EQ.0 .OR. N.EQ.0 ) THEN
        IF( LWORK.GE.1 )
$           WORK( 1 ) = ONE
        RETURN
    END IF
*
*   Get machine constants
*
    EPS = DLAMCH( 'P' )
    SMLNUM = SQRT( DLAMCH( 'S' ) ) / EPS
    BIGNUM = ONE / SMLNUM
*
*   Scale A if max element outside range [SMLNUM,BIGNUM]
*
    ANRM = DLANGE( 'M', M, N, A, LDA, DUM )
    ISCL = 0
    IF( ANRM.GT.ZERO .AND. ANRM.LT.SMLNUM ) THEN
        ISCL = 1
        CALL DLASCL( 'G', 0, 0, ANRM, SMLNUM, M, N, A, LDA, IERR )
    ELSE IF( ANRM.GT.BIGNUM ) THEN
        ISCL = 1
        CALL DLASCL( 'G', 0, 0, ANRM, BIGNUM, M, N, A, LDA, IERR )
    END IF
*
    IF( M.GE.N ) THEN
*
*       A has at least as many rows as columns. If A has sufficiently
*       more rows than columns, first reduce using the QR
*       decomposition (if sufficient workspace available)
*
        IF( M.GE.MNTHR ) THEN
*
*           IF( WNTUN ) THEN
*
*               Path 1 (M much larger than N, JOBU='N')
*               No left singular vectors to be computed
*
*               ITAU = 1
*               IWORK = ITAU + N
*
*           Compute A=Q*R
*           (Workspace: need 2*N, prefer N+N*NB)

```

```

*
*      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ), WORK( IWORK ),
$          LWORK-IWORK+1, IERR )
*
*      Zero out below R
*
*      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ), LDA )
      IE = 1
      ITAUQ = IE + N
      ITAUP = ITAUQ + N
      IWORK = ITAUP + N
*
*      Bidiagonalize R in A
*      (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
*      CALL DGBERD( N, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$          WORK( ITAUP ), WORK( IWORK ), LWORK-IWORK+1,
$          IERR )
      NCVT = 0
      IF( WNTVO .OR. WNTVAS ) THEN
*
*          If right singular vectors desired, generate P'.
*          (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*          CALL DORGBR( 'P', N, N, N, A, LDA, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
          NCVT = N
      END IF
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing right
*      singular vectors of A in A if desired
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', N, NCVT, 0, 0, S, WORK( IE ), A, LDA,
$          DUM, 1, DUM, 1, WORK( IWORK ), INFO )
*
*      If right singular vectors desired in VT, copy them there
*
*      IF( WNTVAS )
$          CALL DLACPY( 'F', N, N, A, LDA, VT, LDVT )
*
*      ELSE IF( WNTUO .AND. WNTVN ) THEN
*
*          Path 2 (M much larger than N, JOBU='O', JOBVT='N')
*          N left singular vectors to be overwritten on A and
*          no right singular vectors to be computed
*
*          IF( LWORK.GE.N*N+MAX( 4*N, BDSPAC ) ) THEN

```

```

*          Sufficient workspace for a fast algorithm
*
*          IR = 1
*          IF( LWORK.GE.MAX( WRKBL, LDA*N+N )+LDA*N ) THEN
*
*              WORK(IU) is LDA by N, WORK(IR) is LDA by N
*
*              LDWRKU = LDA
*              LDWRKR = LDA
*          ELSE IF( LWORK.GE.MAX( WRKBL, LDA*N+N )+N*N ) THEN
*
*              WORK(IU) is LDA by N, WORK(IR) is N by N
*
*              LDWRKU = LDA
*              LDWRKR = N
*          ELSE
*
*              WORK(IU) is LDWRKU by N, WORK(IR) is N by N
*
*              LDWRKU = ( LWORK-N*N-N ) / N
*              LDWRKR = N
*          END IF
*          ITAU = IR + LDWRKR*N
*          IWORK = ITAU + N
*
*          Compute A=Q*R
*          (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*          CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Copy R to WORK(IR) and zero out below it
*
*          CALL DLACPY( 'U', N, N, A, LDA, WORK( IR ), LDWRKR )
*          CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, WORK( IR+1 ),
$              LDWRKR )
*
*          Generate Q in A
*          (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*          CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          IE = ITAU
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          IWORK = ITAUP + N
*
*          Bidiagonalize R in WORK(IR)
*          (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)

```

```

      CALL DGEBRD( N, N, WORK( IR ), LDWRKR, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Generate left vectors bidiagonalizing R
*      (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
      CALL DORGBR( 'Q', N, N, N, WORK( IR ), LDWRKR,
$           WORK( ITAUQ ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of R in WORK(IR)
*      (Workspace: need N*N+BDSPAC)
*
      CALL DBDSQR( 'U', N, 0, N, 0, S, WORK( IE ), DUM, 1,
$           WORK( IR ), LDWRKR, DUM, 1,
$           WORK( IWORK ), INFO )
      IU = IE + N
*
*      Multiply Q in A by left singular vectors of R in
*      WORK(IR), storing result in WORK(IU) and copying to A
*      (Workspace: need N*N+2*N, prefer N*N+M*N+N)
*
      DO 10 I = 1, M, LDWRKU
          CHUNK = MIN( M-I+1, LDWRKU )
          CALL DGEMM( 'N', 'N', CHUNK, N, N, ONE, A( I, 1 ),
$               LDA, WORK( IR ), LDWRKR, ZERO,
$               WORK( IU ), LDWRKU )
          CALL DLACPY( 'F', CHUNK, N, WORK( IU ), LDWRKU,
$               A( I, 1 ), LDA )
10      CONTINUE
*
      ELSE
*
*      Insufficient workspace for a fast algorithm
*
*      IE = 1
*      ITAUQ = IE + N
*      ITAUP = ITAUQ + N
*      IWORK = ITAUP + N
*
*      Bidiagonalize A
*      (Workspace: need 3*N+M, prefer 3*N+(M+N)*NB)
*
      CALL DGEBRD( M, N, A, LDA, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*

```

```

*          Generate left vectors bidiagonalizing A
*          (Workspace: need 4*N, prefer 3*N+N*NB)
*
*          CALL DORGBR( 'Q', M, N, N, A, LDA, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IWORK = IE + N
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of A in A
*          (Workspace: need BDSPAC)
*
*          CALL DBDSQR( 'U', N, 0, M, 0, S, WORK( IE ), DUM, 1,
$              A, LDA, DUM, 1, WORK( IWORK ), INFO )
*
*          END IF
*
*          ELSE IF( WNTUO .AND. WNTVAS ) THEN
*
*          Path 3 (M much larger than N, JOBU='O', JOBVT='S' or 'A')
*          N left singular vectors to be overwritten on A and
*          N right singular vectors to be computed in VT
*
*          IF( LWORK.GE.N*N+MAX( 4*N, BDSPAC ) ) THEN
*
*          Sufficient workspace for a fast algorithm
*
*          IR = 1
*          IF( LWORK.GE.MAX( WRKBL, LDA*N+N )+LDA*N ) THEN
*
*          WORK(IU) is LDA by N and WORK(IR) is LDA by N
*
*          LDWRKU = LDA
*          LDWRKR = LDA
*          ELSE IF( LWORK.GE.MAX( WRKBL, LDA*N+N )+N*N ) THEN
*
*          WORK(IU) is LDA by N and WORK(IR) is N by N
*
*          LDWRKU = LDA
*          LDWRKR = N
*          ELSE
*
*          WORK(IU) is LDWRKU by N and WORK(IR) is N by N
*
*          LDWRKU = ( LWORK-N*N-N ) / N
*          LDWRKR = N
*          END IF
*          ITAU = IR + LDWRKR*N
*          IWORK = ITAU + N
*
*          Compute A=Q*R

```

```

*          (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
*          CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Copy R to VT, zeroing out below it
*
*          CALL DLACPY( 'U', N, N, A, LDA, VT, LDVT )
*          CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, VT( 2, 1 ),
$              LDVT )
*
*          Generate Q in A
*          (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
*          CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          IWORK = ITAUP + N
*
*          Bidiagonalize R in VT, copying result to WORK(IR)
*          (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
*          CALL DGEBRD( N, N, VT, LDVT, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'L', N, N, VT, LDVT, WORK( IR ), LDWRKR )
*
*          Generate left vectors bidiagonalizing R in WORK(IR)
*          (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
*          CALL DORGBR( 'Q', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUQ ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate right vectors bidiagonalizing R in VT
*          (Workspace: need N*N+4*N-1, prefer N*N+3*N+(N-1)*NB)
*
*          CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IWORK = IE + N
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of R in WORK(IR) and computing right
*          singular vectors of R in VT
*          (Workspace: need N*N+BDSPAC)
*
*          CALL DBDSQR( 'U', N, N, N, 0, S, WORK( IE ), VT, LDVT,
$              WORK( IR ), LDWRKR, DUM, 1,

```

```

$                                WORK( IWORK ), INFO )
IU = IE + N
*
*      Multiply Q in A by left singular vectors of R in
*      WORK(IR), storing result in WORK(IU) and copying to A
*      (Workspace: need N*N+2*N, prefer N*N+M*N+N)
*
DO 20 I = 1, M, LDWRKU
    CHUNK = MIN( M-I+1, LDWRKU )
    CALL DGEMM( 'N', 'N', CHUNK, N, N, ONE, A( I, 1 ),
$              LDA, WORK( IR ), LDWRKR, ZERO,
$              WORK( IU ), LDWRKU )
    CALL DLACPY( 'F', CHUNK, N, WORK( IU ), LDWRKU,
$              A( I, 1 ), LDA )
20    CONTINUE
*
ELSE
*
*      Insufficient workspace for a fast algorithm
*
ITAU = 1
IWORK = ITAU + N
*
*      Compute A=Q*R
*      (Workspace: need 2*N, prefer N*N+NB)
*
CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy R to VT, zeroing out below it
*
CALL DLACPY( 'U', N, N, A, LDA, VT, LDVT )
CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, VT( 2, 1 ),
$           LDVT )
*
*      Generate Q in A
*      (Workspace: need 2*N, prefer N*N+NB)
*
CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
IE = ITAU
ITAUQ = IE + N
ITAUP = ITAUQ + N
IWORK = ITAUP + N
*
*      Bidiagonalize R in VT
*      (Workspace: need 4*N, prefer 3*N+2*N+NB)
*
CALL DGEBRD( N, N, VT, LDVT, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),

```



```

$      WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Multiply Q in A by left vectors bidiagonalizing R
*      (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
*      CALL DORMBR( 'Q', 'R', 'N', M, N, N, VT, LDVT,
$              WORK( ITAUQ ), A, LDA, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*      Generate right vectors bidiagonalizing R in VT
*      (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*      CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in A and computing right
*      singular vectors of A in VT
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', N, N, M, 0, S, WORK( IE ), VT, LDVT,
$              A, LDA, DUM, 1, WORK( IWORK ), INFO )
*
*      END IF
*
*      ELSE IF( WNTUS ) THEN
*
*      IF( WNTVN ) THEN
*
*      Path 4 (M much larger than N, JOBU='S', JOBVT='N')
*      N left singular vectors to be computed in U and
*      no right singular vectors to be computed
*
*      IF( LWORK.GE.N*N+MAX( 4*N, BDSPAC ) ) THEN
*
*      Sufficient workspace for a fast algorithm
*
*      IR = 1
*      IF( LWORK.GE.WRKBL+LDA*N ) THEN
*
*      WORK(IR) is LDA by N
*
*      LDWRKR = LDA
*      ELSE
*
*      WORK(IR) is N by N
*
*      LDWRKR = N
*      END IF

```

```

      ITAU = IR + LDWRKR*N
      IWORK = ITAU + N

*
*      Compute A=Q*R
*      (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy R to WORK(IR), zeroing out below it
*
      CALL DLACPY( 'U', N, N, A, LDA, WORK( IR ),
$                LDWRKR )
      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$                WORK( IR+1 ), LDWRKR )
*
*      Generate Q in A
*      (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
      CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$                WORK( IWORK ), LWORK-IWORK+1, IERR )
      IE = ITAU
      ITAUQ = IE + N
      ITAUP = ITAUQ + N
      IWORK = ITAUP + N
*
*      Bidiagonalize R in WORK(IR)
*      (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
      CALL DGEBRD( N, N, WORK( IR ), LDWRKR, S,
$                WORK( IE ), WORK( ITAUQ ),
$                WORK( ITAUP ), WORK( IWORK ),
$                LWORK-IWORK+1, IERR )
*
*      Generate left vectors bidiagonalizing R in WORK(IR)
*      (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
      CALL DORGBR( 'Q', N, N, N, WORK( IR ), LDWRKR,
$                WORK( ITAUQ ), WORK( IWORK ),
$                LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of R in WORK(IR)
*      (Workspace: need N*N+BDSPAC)
*
      CALL DBDSQR( 'U', N, 0, N, 0, S, WORK( IE ), DUM,
$                1, WORK( IR ), LDWRKR, DUM, 1,
$                WORK( IWORK ), INFO )
*

```

```

*          Multiply Q in A by left singular vectors of R in
*          WORK(IR), storing result in U
*          (Workspace: need N*N)
*
*          CALL DGEMM( 'N', 'N', M, N, N, ONE, A, LDA,
$              WORK( IR ), LDWRKR, ZERO, U, LDU )
*
*          ELSE
*
*          Insufficient workspace for a fast algorithm
*
*          ITAU = 1
*          IWORK = ITAU + N
*
*          Compute A=Q*R, copying result to U
*          (Workspace: need 2*N, prefer N+N*NB)
*
*          CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*          Generate Q in U
*          (Workspace: need 2*N, prefer N+N*NB)
*
*          CALL DORGQR( M, N, N, U, LDU, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          IWORK = ITAUP + N
*
*          Zero out below R in A
*
*          CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ),
$              LDA )
*
*          Bidiagonalize R in A
*          (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
*          CALL DGEBRD( N, N, A, LDA, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Multiply Q in U by left vectors bidiagonalizing R
*          (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
*          CALL DORMBR( 'Q', 'R', 'N', M, N, N, A, LDA,
$              WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*          IWORK = IE + N

```

```

*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in U
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', N, 0, M, 0, S, WORK( IE ), DUM,
$          1, U, LDU, DUM, 1, WORK( IWORK ),
$          INFO )
*
*      END IF
*
*      ELSE IF( WNTVO ) THEN
*
*      Path 5 (M much larger than N, JOBU='S', JOBVT='O')
*      N left singular vectors to be computed in U and
*      N right singular vectors to be overwritten on A
*
*      IF( LWORK.GE.2*N*N+MAX( 4*N, BDSPAC ) ) THEN
*
*      Sufficient workspace for a fast algorithm
*
*      IU = 1
*      IF( LWORK.GE.WRKBL+2*LDA*N ) THEN
*
*      WORK(IU) is LDA by N and WORK(IR) is LDA by N
*
*      LDWRKU = LDA
*      IR = IU + LDWRKU*N
*      LDWRKR = LDA
*      ELSE IF( LWORK.GE.WRKBL+( LDA+N )*N ) THEN
*
*      WORK(IU) is LDA by N and WORK(IR) is N by N
*
*      LDWRKU = LDA
*      IR = IU + LDWRKU*N
*      LDWRKR = N
*      ELSE
*
*      WORK(IU) is N by N and WORK(IR) is N by N
*
*      LDWRKU = N
*      IR = IU + LDWRKU*N
*      LDWRKR = N
*      END IF
*      ITAU = IR + LDWRKR*N
*      IWORK = ITAU + N
*
*      Compute A=Q*R
*      (Workspace: need 2*N*N+2*N, prefer 2*N*N+N+N*NB)
*

```

```

      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy R to WORK(IU), zeroing out below it
*
      CALL DLACPY( 'U', N, N, A, LDA, WORK( IU ),
$              LDWRKU )
      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$              WORK( IU+1 ), LDWRKU )
*
*      Generate Q in A
*      (Workspace: need 2*N*N+2*N, prefer 2*N*N+N*N*NB)
*
      CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
      IE = ITAU
      ITAUQ = IE + N
      ITAUP = ITAUQ + N
      IWORK = ITAUP + N
*
*      Bidiagonalize R in WORK(IU), copying result to
*      WORK(IR)
*      (Workspace: need 2*N*N+4*N,
*      prefer 2*N*N+3*N+2*N*NB)
*
      CALL DGEBRD( N, N, WORK( IU ), LDWRKU, S,
$              WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
      CALL DLACPY( 'U', N, N, WORK( IU ), LDWRKU,
$              WORK( IR ), LDWRKR )
*
*      Generate left bidiagonalizing vectors in WORK(IU)
*      (Workspace: need 2*N*N+4*N, prefer 2*N*N+3*N+N*NB)
*
      CALL DORGBR( 'Q', N, N, N, WORK( IU ), LDWRKU,
$              WORK( ITAUQ ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*      Generate right bidiagonalizing vectors in WORK(IR)
*      (Workspace: need 2*N*N+4*N-1,
*      prefer 2*N*N+3*N+(N-1)*NB)
*
      CALL DORGBR( 'P', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of R in WORK(IU) and computing

```

```

*      right singular vectors of R in WORK(IR)
*      (Workspace: need 2*N*N+BDSPAC)
*
*      CALL DBDSQR( 'U', N, N, N, 0, S, WORK( IE ),
$           WORK( IR ), LDWRKR, WORK( IU ),
$           LDWRKU, DUM, 1, WORK( IWORK ), INFO )
*
*      Multiply Q in A by left singular vectors of R in
*      WORK(IU), storing result in U
*      (Workspace: need N*N)
*
*      CALL DGEMM( 'N', 'N', M, N, N, ONE, A, LDA,
$           WORK( IU ), LDWRKU, ZERO, U, LDU )
*
*      Copy right singular vectors of R to A
*      (Workspace: need N*N)
*
*      CALL DLACPY( 'F', N, N, WORK( IR ), LDWRKR, A,
$           LDA )
*
*      ELSE
*
*      Insufficient workspace for a fast algorithm
*
*      ITAU = 1
*      IWORK = ITAU + N
*
*      Compute A=Q*R, copying result to U
*      (Workspace: need 2*N, prefer N+N*NB)
*
*      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*      CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*      Generate Q in U
*      (Workspace: need 2*N, prefer N+N*NB)
*
*      CALL DORGQR( M, N, N, U, LDU, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IE = ITAU
*      ITAUQ = IE + N
*      ITAUP = ITAUQ + N
*      IWORK = ITAUP + N
*
*      Zero out below R in A
*
*      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ),
$           LDA )
*
*      Bidiagonalize R in A

```

```

*          (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
*          CALL DGEBRD( N, N, A, LDA, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Multiply Q in U by left vectors bidiagonalizing R
*          (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
*          CALL DORMBR( 'Q', 'R', 'N', M, N, N, A, LDA,
$              WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate right vectors bidiagonalizing R in A
*          (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*          CALL DORGBR( 'P', N, N, N, A, LDA, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IWORK = IE + N
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of A in U and computing right
*          singular vectors of A in A
*          (Workspace: need BDSPAC)
*
*          CALL DBDSQR( 'U', N, N, M, 0, S, WORK( IE ), A,
$              LDA, U, LDU, DUM, 1, WORK( IWORK ),
$              INFO )
*
*          END IF
*
*          ELSE IF( WNTVAS ) THEN
*
*          Path 6 (M much larger than N, JOBU='S', JOBVT='S'
*              or 'A')
*          N left singular vectors to be computed in U and
*          N right singular vectors to be computed in VT
*
*          IF( LWORK.GE.N*N+MAX( 4*N, BDSPAC ) ) THEN
*
*          Sufficient workspace for a fast algorithm
*
*          IU = 1
*          IF( LWORK.GE.WRKBL+LDA*N ) THEN
*
*          WORK(IU) is LDA by N
*
*          LDWRKU = LDA
*          ELSE

```

```

*          WORK(IU) is N by N
*
*          LDWRKU = N
*          END IF
*          ITAU = IU + LDWRKU*N
*          IWORK = ITAU + N
*
*          Compute A=Q*R
*          (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*          CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Copy R to WORK(IU), zeroing out below it
*
*          CALL DLACPY( 'U', N, N, A, LDA, WORK( IU ),
$                   LDWRKU )
*          CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$                   WORK( IU+1 ), LDWRKU )
*
*          Generate Q in A
*          (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
*          CALL DORGQR( M, N, N, A, LDA, WORK( ITAU ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          IWORK = ITAUP + N
*
*          Bidiagonalize R in WORK(IU), copying result to VT
*          (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
*          CALL DGEBRD( N, N, WORK( IU ), LDWRKU, S,
$                   WORK( IE ), WORK( ITAUQ ),
$                   WORK( ITAUP ), WORK( IWORK ),
$                   LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'U', N, N, WORK( IU ), LDWRKU, VT,
$                   LDVT )
*
*          Generate left bidiagonalizing vectors in WORK(IU)
*          (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
*          CALL DORGBR( 'Q', N, N, N, WORK( IU ), LDWRKU,
$                   WORK( ITAUQ ), WORK( IWORK ),
$                   LWORK-IWORK+1, IERR )
*
*          Generate right bidiagonalizing vectors in VT
*          (Workspace: need N*N+4*N-1,
*                   prefer N*N+3*N+(N-1)*NB)

```



```

*
*      CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of R in WORK(IU) and computing
*      right singular vectors of R in VT
*      (Workspace: need N*N+BDSPAC)
*
*      CALL DBDSQR( 'U', N, N, N, 0, S, WORK( IE ), VT,
$          LDVT, WORK( IU ), LDWRKU, DUM, 1,
$          WORK( IWORK ), INFO )
*
*      Multiply Q in A by left singular vectors of R in
*      WORK(IU), storing result in U
*      (Workspace: need N*N)
*
*      CALL DGEMM( 'N', 'N', M, N, N, ONE, A, LDA,
$          WORK( IU ), LDWRKU, ZERO, U, LDU )
*
*      ELSE
*
*      Insufficient workspace for a fast algorithm
*
*      ITAU = 1
*      IWORK = ITAU + N
*
*      Compute A=Q*R, copying result to U
*      (Workspace: need 2*N, prefer N+N*NB)
*
*      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*      CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*      Generate Q in U
*      (Workspace: need 2*N, prefer N+N*NB)
*
*      CALL DORGQR( M, N, N, U, LDU, WORK( ITAU ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy R to VT, zeroing out below it
*
*      CALL DLACPY( 'U', N, N, A, LDA, VT, LDVT )
*      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, VT( 2, 1 ),
$          LDVT )
*
*      IE = ITAU
*      ITAUQ = IE + N
*      ITAUP = ITAUQ + N
*      IWORK = ITAUP + N

```

```

*
*      Bidiagonalize R in VT
*      (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
*      CALL DGEBRD( N, N, VT, LDVT, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Multiply Q in U by left bidiagonalizing vectors
*      in VT
*      (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
*      CALL DORMBR( 'Q', 'R', 'N', M, N, N, VT, LDVT,
$              WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*      Generate right bidiagonalizing vectors in VT
*      (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*      CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in U and computing right
*      singular vectors of A in VT
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', N, N, M, 0, S, WORK( IE ), VT,
$              LDVT, U, LDU, DUM, 1, WORK( IWORK ),
$              INFO )
*
*      END IF
*
*      END IF
*
*      ELSE IF( WNTUA ) THEN
*
*          IF( WNTVN ) THEN
*
*              Path 7 (M much larger than N, JOBU='A', JOBVT='N')
*              M left singular vectors to be computed in U and
*              no right singular vectors to be computed
*
*              IF( LWORK.GE.N*N+MAX( N+M, 4*N, BDSPAC ) ) THEN
*
*                  Sufficient workspace for a fast algorithm
*
*                  IR = 1
*                  IF( LWORK.GE.WRKBL+LDA*N ) THEN

```

```

*
*          WORK(IR) is LDA by N
*
*          LDWRKR = LDA
*        ELSE
*
*          WORK(IR) is N by N
*
*          LDWRKR = N
*        END IF
*        ITAU = IR + LDWRKR*N
*        IWORK = ITAU + N
*
*        Compute A=Q*R, copying result to U
*        (Workspace: need N*N+2*N, prefer N*N+N+N*NB)
*
*        CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*        CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*        Copy R to WORK(IR), zeroing out below it
*
*        CALL DLACPY( 'U', N, N, A, LDA, WORK( IR ),
$              LDWRKR )
*        CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$              WORK( IR+1 ), LDWRKR )
*
*        Generate Q in U
*        (Workspace: need N*N+N+M, prefer N*N+N+M*NB)
*
*        CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*        IE = ITAU
*        ITAUQ = IE + N
*        ITAUP = ITAUQ + N
*        IWORK = ITAUP + N
*
*        Bidiagonalize R in WORK(IR)
*        (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
*        CALL DGEBRD( N, N, WORK( IR ), LDWRKR, S,
$              WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*        Generate left bidiagonalizing vectors in WORK(IR)
*        (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
*        CALL DORGBR( 'Q', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUQ ), WORK( IWORK ),

```

```

$                                LWORK-IWORK+1, IERR )
IWORK = IE + N
*
* Perform bidiagonal QR iteration, computing left
* singular vectors of R in WORK(IR)
* (Workspace: need N*N+BDSPAC)
*
CALL DBDSQR( 'U', N, 0, N, 0, S, WORK( IE ), DUM,
$           1, WORK( IR ), LDWRKR, DUM, 1,
$           WORK( IWORK ), INFO )
*
* Multiply Q in U by left singular vectors of R in
* WORK(IR), storing result in A
* (Workspace: need N*N)
*
CALL DGEMM( 'N', 'N', M, N, N, ONE, U, LDU,
$           WORK( IR ), LDWRKR, ZERO, A, LDA )
*
* Copy left singular vectors of A from A to U
*
CALL DLACPY( 'F', M, N, A, LDA, U, LDU )
*
ELSE
*
* Insufficient workspace for a fast algorithm
*
*
* ITAU = 1
* IWORK = ITAU + N
*
* Compute A=Q*R, copying result to U
* (Workspace: need 2*N, prefer N+N*NB)
*
CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
* Generate Q in U
* (Workspace: need N+M, prefer N+M*NB)
*
CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
* IE = ITAU
* ITAUQ = IE + N
* ITAUP = ITAUQ + N
* IWORK = ITAUP + N
*
* Zero out below R in A
*
CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ),
$           LDA )

```

```

*
*      Bidiagonalize R in A
*      (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
      CALL DGEBRD( N, N, A, LDA, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Multiply Q in U by left bidiagonalizing vectors
*      in A
*      (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
      CALL DORMBR( 'Q', 'R', 'N', M, N, N, A, LDA,
$              WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in U
*      (Workspace: need BDSPAC)
*
      CALL DBDSQR( 'U', N, 0, M, 0, S, WORK( IE ), DUM,
$              1, U, LDU, DUM, 1, WORK( IWORK ),
$              INFO )
*
      END IF
*
      ELSE IF( WNTVO ) THEN
*
*      Path 8 (M much larger than N, JOBU='A', JOBVT='O')
*      M left singular vectors to be computed in U and
*      N right singular vectors to be overwritten on A
*
      IF( LWORK.GE.2*N*N+MAX( N+M, 4*N, BDSPAC ) ) THEN
*
*      Sufficient workspace for a fast algorithm
*
*      IU = 1
*      IF( LWORK.GE.WRKBL+2*LDA*N ) THEN
*
*      WORK(IU) is LDA by N and WORK(IR) is LDA by N
*
*      LDWRKU = LDA
*      IR = IU + LDWRKU*N
*      LDWRKR = LDA
*      ELSE IF( LWORK.GE.WRKBL+( LDA+N )*N ) THEN
*
*      WORK(IU) is LDA by N and WORK(IR) is N by N
*
*      LDWRKU = LDA

```

```

        IR = IU + LDWRKU*N
        LDWRKR = N
ELSE
*
*       WORK(IU) is N by N and WORK(IR) is N by N
*
        LDWRKU = N
        IR = IU + LDWRKU*N
        LDWRKR = N
END IF
ITAU = IR + LDWRKR*N
IWORK = ITAU + N
*
*       Compute A=Q*R, copying result to U
*       (Workspace: need 2*N*N+2*N, prefer 2*N*N+N+N*N)
*
        CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
        CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*       Generate Q in U
*       (Workspace: need 2*N*N+N*M, prefer 2*N*N+N*M*N)
*
        CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       Copy R to WORK(IU), zeroing out below it
*
        CALL DLACPY( 'U', N, N, A, LDA, WORK( IU ),
$                   LDWRKU )
        CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$                   WORK( IU+1 ), LDWRKU )
        IE = ITAU
        ITAUQ = IE + N
        ITAUP = ITAUQ + N
        IWORK = ITAUP + N
*
*       Bidiagonalize R in WORK(IU), copying result to
*       WORK(IR)
*       (Workspace: need 2*N*N+4*N,
*       prefer 2*N*N+3*N+2*N*N)
*
        CALL DGEBRD( N, N, WORK( IU ), LDWRKU, S,
$                   WORK( IE ), WORK( ITAUQ ),
$                   WORK( ITAUP ), WORK( IWORK ),
$                   LWORK-IWORK+1, IERR )
        CALL DLACPY( 'U', N, N, WORK( IU ), LDWRKU,
$                   WORK( IR ), LDWRKR )
*
*       Generate left bidiagonalizing vectors in WORK(IU)

```

```

*          (Workspace: need 2*N*N+4*N, prefer 2*N*N+3*N+N*N)
*
*          CALL DORGBR( 'Q', N, N, N, WORK( IU ), LDWRKU,
$              WORK( ITAUQ ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate right bidiagonalizing vectors in WORK(IR)
*          (Workspace: need 2*N*N+4*N-1,
*              prefer 2*N*N+3*N+(N-1)*N)
*
*          CALL DORGBR( 'P', N, N, N, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*          IWORK = IE + N
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of R in WORK(IU) and computing
*          right singular vectors of R in WORK(IR)
*          (Workspace: need 2*N*N+BDSPAC)
*
*          CALL DBDSQR( 'U', N, N, N, 0, S, WORK( IE ),
$              WORK( IR ), LDWRKR, WORK( IU ),
$              LDWRKU, DUM, 1, WORK( IWORK ), INFO )
*
*          Multiply Q in U by left singular vectors of R in
*          WORK(IU), storing result in A
*          (Workspace: need N*N)
*
*          CALL DGEMM( 'N', 'N', M, N, N, ONE, U, LDU,
$              WORK( IU ), LDWRKU, ZERO, A, LDA )
*
*          Copy left singular vectors of A from A to U
*
*          CALL DLACPY( 'F', M, N, A, LDA, U, LDU )
*
*          Copy right singular vectors of R from WORK(IR) to A
*
*          CALL DLACPY( 'F', N, N, WORK( IR ), LDWRKR, A,
$              LDA )
*
*      ELSE
*
*          Insufficient workspace for a fast algorithm
*
*          ITAU = 1
*          IWORK = ITAU + N
*
*          Compute A=Q*R, copying result to U
*          (Workspace: need 2*N, prefer N+N*N)
*

```

```

CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*   Generate Q in U
*   (Workspace: need N+M, prefer N+M*NB)
*
CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
IE = ITAU
ITAUQ = IE + N
ITAUP = ITAUQ + N
IWORK = ITAUP + N
*
*   Zero out below R in A
*
CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, A( 2, 1 ),
$           LDA )
*
*   Bidiagonalize R in A
*   (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
CALL DGEBRD( N, N, A, LDA, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*   Multiply Q in U by left bidiagonalizing vectors
*   in A
*   (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
CALL DORMBR( 'Q', 'R', 'N', M, N, N, A, LDA,
$           WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
*
*   Generate right bidiagonalizing vectors in A
*   (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
CALL DORGBR( 'P', N, N, N, A, LDA, WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
IWORK = IE + N
*
*   Perform bidiagonal QR iteration, computing left
*   singular vectors of A in U and computing right
*   singular vectors of A in A
*   (Workspace: need BDSPAC)
*
CALL DBDSQR( 'U', N, N, M, 0, S, WORK( IE ), A,
$           LDA, U, LDU, DUM, 1, WORK( IWORK ),
$           INFO )
*

```



```

      END IF
*
      ELSE IF( WNTVAS ) THEN
*
*       Path 9 (M much larger than N, JOBU='A', JOBVT='S'
*       or 'A')
*       M left singular vectors to be computed in U and
*       N right singular vectors to be computed in VT
*
      IF( LWORK.GE.N*N+MAX( N+M, 4*N, BDSPAC ) ) THEN
*
*       Sufficient workspace for a fast algorithm
*
        IU = 1
        IF( LWORK.GE.WRKBL+LDA*N ) THEN
*
*         WORK(IU) is LDA by N
*
          LDWRKU = LDA
        ELSE
*
*         WORK(IU) is N by N
*
          LDWRKU = N
        END IF
        ITAU = IU + LDWRKU*N
        IWORK = ITAU + N
*
*       Compute A=Q*R, copying result to U
*       (Workspace: need N*N+2*N, prefer N*N+N*N*NB)
*
        CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
        CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*       Generate Q in U
*       (Workspace: need N*N+N*M, prefer N*N+N*M*NB)
*
        CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       Copy R to WORK(IU), zeroing out below it
*
        CALL DLACPY( 'U', N, N, A, LDA, WORK( IU ),
$              LDWRKU )
        CALL DLASET( 'L', N-1, N-1, ZERO, ZERO,
$              WORK( IU+1 ), LDWRKU )
        IE = ITAU
        ITAUQ = IE + N
        ITAUP = ITAUQ + N

```

```

      IWORK = ITAUP + N
*
*      Bidiagonalize R in WORK(IU), copying result to VT
*      (Workspace: need N*N+4*N, prefer N*N+3*N+2*N*NB)
*
      CALL DGEBRD( N, N, WORK( IU ), LDWRKU, S,
$                WORK( IE ), WORK( ITAUQ ),
$                WORK( ITAUP ), WORK( IWORK ),
$                LWORK-IWORK+1, IERR )
      CALL DLACPY( 'U', N, N, WORK( IU ), LDWRKU, VT,
$                LDVT )
*
*      Generate left bidiagonalizing vectors in WORK(IU)
*      (Workspace: need N*N+4*N, prefer N*N+3*N+N*NB)
*
      CALL DORGBR( 'Q', N, N, N, N, WORK( IU ), LDWRKU,
$                WORK( ITAUQ ), WORK( IWORK ),
$                LWORK-IWORK+1, IERR )
*
*      Generate right bidiagonalizing vectors in VT
*      (Workspace: need N*N+4*N-1,
*                prefer N*N+3*N+(N-1)*NB)
*
      CALL DORGBR( 'P', N, N, N, N, VT, LDVT, WORK( ITAUP ),
$                WORK( IWORK ), LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of R in WORK(IU) and computing
*      right singular vectors of R in VT
*      (Workspace: need N*N+BDSPAC)
*
      CALL DBDSQR( 'U', N, N, N, N, 0, S, WORK( IE ), VT,
$                LDVT, WORK( IU ), LDWRKU, DUM, 1,
$                WORK( IWORK ), INFO )
*
*      Multiply Q in U by left singular vectors of R in
*      WORK(IU), storing result in A
*      (Workspace: need N*N)
*
      CALL DGEMM( 'N', 'N', M, N, N, ONE, U, LDU,
$                WORK( IU ), LDWRKU, ZERO, A, LDA )
*
*      Copy left singular vectors of A from A to U
*
      CALL DLACPY( 'F', M, N, A, LDA, U, LDU )
*
ELSE
*
*      Insufficient workspace for a fast algorithm

```

```

*
      ITAU = 1
      IWORK = ITAU + N
*
*      Compute A=Q*R, copying result to U
*      (Workspace: need 2*N, prefer N+N*NB)
*
      CALL DGEQRF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
      CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*
*      Generate Q in U
*      (Workspace: need N+M, prefer N+M*NB)
*
      CALL DORGQR( M, M, N, U, LDU, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy R from A to VT, zeroing out below it
*
      CALL DLACPY( 'U', N, N, A, LDA, VT, LDVT )
      CALL DLASET( 'L', N-1, N-1, ZERO, ZERO, VT( 2, 1 ),
$              LDVT )
      IE = ITAU
      ITAUQ = IE + N
      ITAUP = ITAUQ + N
      IWORK = ITAUP + N
*
*      Bidiagonalize R in VT
*      (Workspace: need 4*N, prefer 3*N+2*N*NB)
*
      CALL DGEBRD( N, N, VT, LDVT, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Multiply Q in U by left bidiagonalizing vectors
*      in VT
*      (Workspace: need 3*N+M, prefer 3*N+M*NB)
*
      CALL DORMBR( 'Q', 'R', 'N', M, N, N, VT, LDVT,
$              WORK( ITAUQ ), U, LDU, WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*      Generate right bidiagonalizing vectors in VT
*      (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
      CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
      IWORK = IE + N
*
*      Perform bidiagonal QR iteration, computing left

```

```

*          singular vectors of A in U and computing right
*          singular vectors of A in VT
*          (Workspace: need BDSPAC)
*
*          CALL DBDSQR( 'U', N, N, M, 0, S, WORK( IE ), VT,
$              LDVT, U, LDU, DUM, 1, WORK( IWORK ),
$              INFO )
*
*          END IF
*
*          END IF
*
*          END IF
*
*      ELSE
*
*          M .LT. MNTHR
*
*          Path 10 (M at least N, but not much larger)
*          Reduce to bidiagonal form without QR decomposition
*
*          IE = 1
*          ITAUQ = IE + N
*          ITAUP = ITAUQ + N
*          IWORK = ITAUP + N
*
*          Bidiagonalize A
*          (Workspace: need 3*N+M, prefer 3*N+(M+N)*NB)
*
*          CALL DGEBRD( M, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ), LWORK-IWORK+1,
$              IERR )
*          IF( WNTUAS ) THEN
*
*              If left singular vectors desired in U, copy result to U
*              and generate left bidiagonalizing vectors in U
*              (Workspace: need 3*N+NCU, prefer 3*N+NCU*NB)
*
*              CALL DLACPY( 'L', M, N, A, LDA, U, LDU )
*              IF( WNTUS )
$                  NCU = N
*              IF( WNTUA )
$                  NCU = M
*              CALL DORGBR( 'Q', M, NCU, N, U, LDU, WORK( ITAUQ ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*          END IF
*          IF( WNTVAS ) THEN
*
*              If right singular vectors desired in VT, copy result to
*              VT and generate right bidiagonalizing vectors in VT

```

```

*          (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*          CALL DLACPY( 'U', N, N, A, LDA, VT, LDVT )
*          CALL DORGBR( 'P', N, N, N, VT, LDVT, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          END IF
*          IF( WNTUO ) THEN
*
*              If left singular vectors desired in A, generate left
*              bidiagonalizing vectors in A
*              (Workspace: need 4*N, prefer 3*N+N*NB)
*
*              CALL DORGBR( 'Q', M, N, N, A, LDA, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          END IF
*          IF( WNTVO ) THEN
*
*              If right singular vectors desired in A, generate right
*              bidiagonalizing vectors in A
*              (Workspace: need 4*N-1, prefer 3*N+(N-1)*NB)
*
*              CALL DORGBR( 'P', N, N, N, A, LDA, WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          END IF
*          IWORK = IE + N
*          IF( WNTUAS .OR. WNTUO )
$              NRU = M
*          IF( WNTUN )
$              NRU = 0
*          IF( WNTVAS .OR. WNTVO )
$              NCVT = N
*          IF( WNTVN )
$              NCVT = 0
*          IF( ( .NOT.WNTUO ) .AND. ( .NOT.WNTVO ) ) THEN
*
*              Perform bidiagonal QR iteration, if desired, computing
*              left singular vectors in U and computing right singular
*              vectors in VT
*              (Workspace: need BDSPAC)
*
*              CALL DBDSQR( 'U', N, NCVT, NRU, 0, S, WORK( IE ), VT,
$              LDVT, U, LDU, DUM, 1, WORK( IWORK ), INFO )
*          ELSE IF( ( .NOT.WNTUO ) .AND. WNTVO ) THEN
*
*              Perform bidiagonal QR iteration, if desired, computing
*              left singular vectors in U and computing right singular
*              vectors in A
*              (Workspace: need BDSPAC)
*
*              CALL DBDSQR( 'U', N, NCVT, NRU, 0, S, WORK( IE ), A, LDA,

```

```

$          U, LDU, DUM, 1, WORK( IWORK ), INFO )
      ELSE
*
*          Perform bidiagonal QR iteration, if desired, computing
*          left singular vectors in A and computing right singular
*          vectors in VT
*          (Workspace: need BDSPAC)
*
*          CALL DBDSQR( 'U', N, NCVT, NRU, 0, S, WORK( IE ), VT,
$          LDVT, A, LDA, DUM, 1, WORK( IWORK ), INFO )
      END IF
*
      END IF
*
      ELSE
*
*          A has more columns than rows. If A has sufficiently more
*          columns than rows, first reduce using the LQ decomposition (if
*          sufficient workspace available)
*
      IF( N.GE.MNTHR ) THEN
*
*          IF( WNTVN ) THEN
*
*              Path 1t(N much larger than M, JOBVT='N')
*              No right singular vectors to be computed
*
*              ITAU = 1
*              IWORK = ITAU + M
*
*              Compute A=L*Q
*              (Workspace: need 2*M, prefer M+M*NB)
*
*              CALL DGELQF( M, N, A, LDA, WORK( ITAU ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*              Zero out above L
*
*              CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ), LDA )
*              IE = 1
*              ITAUQ = IE + M
*              ITAUP = ITAUQ + M
*              IWORK = ITAUP + M
*
*              Bidiagonalize L in A
*              (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*              CALL DGEBRD( M, M, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ), LWORK-IWORK+1,
$              IERR )

```

```

      IF( WNTUO .OR. WNTUAS ) THEN
*
*       If left singular vectors desired, generate Q
*       (Workspace: need 4*M, prefer 3*M+M*N)
*
*       CALL DORGBR( 'Q', M, M, M, A, LDA, WORK( ITAUQ ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
      END IF
      IWORK = IE + M
      NRU = 0
      IF( WNTUO .OR. WNTUAS )
$         NRU = M
*
*       Perform bidiagonal QR iteration, computing left singular
*       vectors of A in A if desired
*       (Workspace: need BDSPAC)
*
*       CALL DBDSQR( 'U', M, 0, NRU, 0, S, WORK( IE ), DUM, 1, A,
$           LDA, DUM, 1, WORK( IWORK ), INFO )
*
*       If left singular vectors desired in U, copy them there
*
*       IF( WNTUAS )
$         CALL DLACPY( 'F', M, M, A, LDA, U, LDU )
*
      ELSE IF( WNTVO .AND. WNTUN ) THEN
*
*       Path 2t(N much larger than M, JOBU='N', JOBVT='O')
*       M right singular vectors to be overwritten on A and
*       no left singular vectors to be computed
*
*       IF( LWORK.GE.M*M+MAX( 4*M, BDSPAC ) ) THEN
*
*       Sufficient workspace for a fast algorithm
*
*       IR = 1
*       IF( LWORK.GE.MAX( WRKBL, LDA*N+M )+LDA*M ) THEN
*
*       WORK(IU) is LDA by N and WORK(IR) is LDA by M
*
*       LDWRKU = LDA
*       CHUNK = N
*       LDWRKR = LDA
*       ELSE IF( LWORK.GE.MAX( WRKBL, LDA*N+M )+M*M ) THEN
*
*       WORK(IU) is LDA by N and WORK(IR) is M by M
*
*       LDWRKU = LDA
*       CHUNK = N
*       LDWRKR = M

```

```

ELSE
*
*      WORK(IU) is M by CHUNK and WORK(IR) is M by M
*
      LDWRKU = M
      CHUNK = ( LWORK-M*M-M ) / M
      LDWRKR = M
END IF
ITAU = IR + LDWRKR*M
IWORK = ITAU + M
*
*      Compute A=L*Q
*      (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
      CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Copy L to WORK(IR) and zero out above it
*
      CALL DLACPY( 'L', M, M, A, LDA, WORK( IR ), LDWRKR )
      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$           WORK( IR+LDWRKR ), LDWRKR )
*
*      Generate Q in A
*      (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
      CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
      IE = ITAU
      ITAUQ = IE + M
      ITAUP = ITAUQ + M
      IWORK = ITAUP + M
*
*      Bidiagonalize L in WORK(IR)
*      (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*Nb)
*
      CALL DGEBRD( M, M, WORK( IR ), LDWRKR, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Generate right vectors bidiagonalizing L
*      (Workspace: need M*M+4*M-1, prefer M*M+3*M+(M-1)*Nb)
*
      CALL DORGBR( 'P', M, M, M, WORK( IR ), LDWRKR,
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing right
*      singular vectors of L in WORK(IR)

```



```

*          (Workspace: need M*M+BDSAPC)
*
*          CALL DBDSQR( 'U', M, M, 0, 0, S, WORK( IE ),
$              WORK( IR ), LDWRKR, DUM, 1, DUM, 1,
$              WORK( IWORK ), INFO )
*          IU = IE + M
*
*          Multiply right singular vectors of L in WORK(IR) by Q
*          in A, storing result in WORK(IU) and copying to A
*          (Workspace: need M*M+2*M, prefer M*M+M*N+M)
*
*          DO 30 I = 1, N, CHUNK
*              BLK = MIN( N-I+1, CHUNK )
*              CALL DGEMM( 'N', 'N', M, BLK, M, ONE, WORK( IR ),
$                  LDWRKR, A( 1, I ), LDA, ZERO,
$                  WORK( IU ), LDWRKU )
*              CALL DLACPY( 'F', M, BLK, WORK( IU ), LDWRKU,
$                  A( 1, I ), LDA )
30          CONTINUE
*
*          ELSE
*
*              Insufficient workspace for a fast algorithm
*
*              IE = 1
*              ITAUQ = IE + M
*              ITAUP = ITAUQ + M
*              IWORK = ITAUP + M
*
*              Bidiagonalize A
*              (Workspace: need 3*M+N, prefer 3*M+(M+N)*NB)
*
*              CALL DGEBRD( M, N, A, LDA, S, WORK( IE ),
$                  WORK( ITAUQ ), WORK( ITAUP ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*              Generate right vectors bidiagonalizing A
*              (Workspace: need 4*M, prefer 3*M+M*N)
*
*              CALL DORGBR( 'P', M, N, M, A, LDA, WORK( ITAUP ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*              IWORK = IE + M
*
*              Perform bidiagonal QR iteration, computing right
*              singular vectors of A in A
*              (Workspace: need BDSAPC)
*
*              CALL DBDSQR( 'L', M, N, 0, 0, S, WORK( IE ), A, LDA,
$                  DUM, 1, DUM, 1, WORK( IWORK ), INFO )
*

```

```

      END IF
*
      ELSE IF( WNTVO .AND. WNTUAS ) THEN
*
*       Path 3t(N much larger than M, JOBU='S' or 'A', JOBVT='O')
*       M right singular vectors to be overwritten on A and
*       M left singular vectors to be computed in U
*
      IF( LWORK.GE.M*M+MAX( 4*M, BDSPAC ) ) THEN
*
*       Sufficient workspace for a fast algorithm
*
      IR = 1
      IF( LWORK.GE.MAX( WRKBL, LDA*N+M )+LDA*M ) THEN
*
*       WORK(IU) is LDA by N and WORK(IR) is LDA by M
*
*       LDWRKU = LDA
*       CHUNK = N
*       LDWRKR = LDA
      ELSE IF( LWORK.GE.MAX( WRKBL, LDA*N+M )+M*M ) THEN
*
*       WORK(IU) is LDA by N and WORK(IR) is M by M
*
*       LDWRKU = LDA
*       CHUNK = N
*       LDWRKR = M
      ELSE
*
*       WORK(IU) is M by CHUNK and WORK(IR) is M by M
*
*       LDWRKU = M
*       CHUNK = ( LWORK-M*M-M ) / M
*       LDWRKR = M
      END IF
      ITAU = IR + LDWRKR*M
      IWORK = ITAU + M
*
*       Compute A=L*Q
*       (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
      CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       Copy L to U, zeroing about above it
*
      CALL DLACPY( 'L', M, M, A, LDA, U, LDU )
      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, U( 1, 2 ),
$                LDU )
*

```

```

*          Generate Q in A
*          (Workspace: need M*M+2*M, prefer M*M+M*M*NB)
*
*          CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M
*          IWORK = ITAUP + M
*
*          Bidiagonalize L in U, copying result to WORK(IR)
*          (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
*          CALL DGEBRD( M, M, U, LDU, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'U', M, M, U, LDU, WORK( IR ), LDWRKR )
*
*          Generate right vectors bidiagonalizing L in WORK(IR)
*          (Workspace: need M*M+4*M-1, prefer M*M+3*M+(M-1)*NB)
*
*          CALL DORGBR( 'P', M, M, M, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate left vectors bidiagonalizing L in U
*          (Workspace: need M*M+4*M, prefer M*M+3*M*M*NB)
*
*          CALL DORGBR( 'Q', M, M, M, U, LDU, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IWORK = IE + M
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of L in U, and computing right
*          singular vectors of L in WORK(IR)
*          (Workspace: need M*M+BDSPAC)
*
*          CALL DBDSQR( 'U', M, M, M, 0, S, WORK( IE ),
$              WORK( IR ), LDWRKR, U, LDU, DUM, 1,
$              WORK( IWORK ), INFO )
*          IU = IE + M
*
*          Multiply right singular vectors of L in WORK(IR) by Q
*          in A, storing result in WORK(IU) and copying to A
*          (Workspace: need M*M+2*M, prefer M*M+M*N+M)
*
*          DO 40 I = 1, N, CHUNK
*              BLK = MIN( N-I+1, CHUNK )
*              CALL DGEMM( 'N', 'N', M, BLK, M, ONE, WORK( IR ),
$                  LDWRKR, A( 1, I ), LDA, ZERO,

```

```

$                                WORK( IU ), LDWRKU )
    CALL DLACPY( 'F', M, BLK, WORK( IU ), LDWRKU,
$                                A( 1, I ), LDA )
40    CONTINUE
*
*    ELSE
*
*    Insufficient workspace for a fast algorithm
*
*    ITAU = 1
*    IWORK = ITAU + M
*
*    Compute A=L*Q
*    (Workspace: need 2*M, prefer M*M*NB)
*
*    CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*    Copy L to U, zeroing out above it
*
*    CALL DLACPY( 'L', M, M, A, LDA, U, LDU )
*    CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, U( 1, 2 ),
$                                LDU )
*
*    Generate Q in A
*    (Workspace: need 2*M, prefer M*M*NB)
*
*    CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*    IE = ITAU
*    ITAUQ = IE + M
*    ITAUP = ITAUQ + M
*    IWORK = ITAUP + M
*
*    Bidiagonalize L in U
*    (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*    CALL DGEBRD( M, M, U, LDU, S, WORK( IE ),
$                                WORK( ITAUQ ), WORK( ITAUP ),
$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*    Multiply right vectors bidiagonalizing L by Q in A
*    (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*    CALL DORMBR( 'P', 'L', 'T', M, N, M, U, LDU,
$                                WORK( ITAUP ), A, LDA, WORK( IWORK ),
$                                LWORK-IWORK+1, IERR )
*
*    Generate left vectors bidiagonalizing L in U
*    (Workspace: need 4*M, prefer 3*M+M*NB)

```

```

*
*      CALL DORGBR( 'Q', M, M, M, U, LDU, WORK( ITAUQ ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in U and computing right
*      singular vectors of A in A
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', M, N, M, 0, S, WORK( IE ), A, LDA,
$          U, LDU, DUM, 1, WORK( IWORK ), INFO )
*
*      END IF
*
*      ELSE IF( WNTVS ) THEN
*
*          IF( WNTUN ) THEN
*
*              Path 4t(N much larger than M, JOBU='N', JOBVT='S')
*              M right singular vectors to be computed in VT and
*              no left singular vectors to be computed
*
*              IF( LWORK.GE.M*M+MAX( 4*M, BDSPAC ) ) THEN
*
*                  Sufficient workspace for a fast algorithm
*
*                  IR = 1
*                  IF( LWORK.GE.WRKBL+LDA*M ) THEN
*
*                      WORK(IR) is LDA by M
*
*                      LDWRKR = LDA
*                  ELSE
*
*                      WORK(IR) is M by M
*
*                      LDWRKR = M
*                  END IF
*                  ITAU = IR + LDWRKR*M
*                  IWORK = ITAU + M
*
*              Compute A=L*Q
*              (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*              CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*              Copy L to WORK(IR), zeroing out above it
*

```

```

CALL DLACPY( 'L', M, M, A, LDA, WORK( IR ),
$           LDWRKR )
CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$           WORK( IR+LDWRKR ), LDWRKR )
*
*       Generate Q in A
*       (Workspace: need M*M+2*M, prefer M*M+M*M*NB)
*
CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
IE = ITAU
ITAUQ = IE + M
ITAUP = ITAUQ + M
IWORK = ITAUP + M
*
*       Bidiagonalize L in WORK(IR)
*       (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
CALL DGEBRD( M, M, WORK( IR ), LDWRKR, S,
$           WORK( IE ), WORK( ITAUQ ),
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
*
*       Generate right vectors bidiagonalizing L in
*       WORK(IR)
*       (Workspace: need M*M+4*M, prefer M*M+3*M+(M-1)*NB)
*
CALL DORGBR( 'P', M, M, M, WORK( IR ), LDWRKR,
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
IWORK = IE + M
*
*       Perform bidiagonal QR iteration, computing right
*       singular vectors of L in WORK(IR)
*       (Workspace: need M*M+BDSPAC)
*
CALL DBDSQR( 'U', M, M, 0, 0, S, WORK( IE ),
$           WORK( IR ), LDWRKR, DUM, 1, DUM, 1,
$           WORK( IWORK ), INFO )
*
*       Multiply right singular vectors of L in WORK(IR) by
*       Q in A, storing result in VT
*       (Workspace: need M*M)
*
CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IR ),
$           LDWRKR, A, LDA, ZERO, VT, LDVT )
*
ELSE
*
*       Insufficient workspace for a fast algorithm

```

```

*
*       ITAU = 1
*       IWORK = ITAU + M
*
*       Compute A=L*Q
*       (Workspace: need 2*M, prefer M+M*NB)
*
*       CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       Copy result to VT
*
*       CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*       Generate Q in VT
*       (Workspace: need 2*M, prefer M+M*NB)
*
*       CALL DORGLQ( M, N, M, VT, LDVT, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       IE = ITAU
*       ITAUQ = IE + M
*       ITAUP = ITAUQ + M
*       IWORK = ITAUP + M
*
*       Zero out above L in A
*
*       CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ),
$           LDA )
*
*       Bidiagonalize L in A
*       (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*       CALL DGEBRD( M, M, A, LDA, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       Multiply right vectors bidiagonalizing L by Q in VT
*       (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*       CALL DORMBR( 'P', 'L', 'T', M, N, M, A, LDA,
$           WORK( ITAUP ), VT, LDVT,
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*       IWORK = IE + M
*
*       Perform bidiagonal QR iteration, computing right
*       singular vectors of A in VT
*       (Workspace: need BDSPAC)
*
*       CALL DBDSQR( 'U', M, N, 0, 0, S, WORK( IE ), VT,
$           LDVT, DUM, 1, DUM, 1, WORK( IWORK ),

```

```

$                                INFO )
*
*                                END IF
*
*                                ELSE IF( WNTUO ) THEN
*
*                                Path 5t(N much larger than M, JOBU='O', JOBVT='S')
*                                M right singular vectors to be computed in VT and
*                                M left singular vectors to be overwritten on A
*
*                                IF( LWORK.GE.2*M*M+MAX( 4*M, BDSPAC ) ) THEN
*
*                                Sufficient workspace for a fast algorithm
*
*                                IU = 1
*                                IF( LWORK.GE.WRKBL+2*LDA*M ) THEN
*
*                                WORK(IU) is LDA by M and WORK(IR) is LDA by M
*
*                                LDWRKU = LDA
*                                IR = IU + LDWRKU*M
*                                LDWRKR = LDA
*                                ELSE IF( LWORK.GE.WRKBL+( LDA+M )*M ) THEN
*
*                                WORK(IU) is LDA by M and WORK(IR) is M by M
*
*                                LDWRKU = LDA
*                                IR = IU + LDWRKU*M
*                                LDWRKR = M
*                                ELSE
*
*                                WORK(IU) is M by M and WORK(IR) is M by M
*
*                                LDWRKU = M
*                                IR = IU + LDWRKU*M
*                                LDWRKR = M
*                                END IF
*                                ITAU = IR + LDWRKR*M
*                                IWORK = ITAU + M
*
*                                Compute A=L*Q
*                                (Workspace: need 2*M*M+2*M, prefer 2*M*M+M+M*NB)
*
*                                CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$                                    WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*                                Copy L to WORK(IU), zeroing out below it
*
*                                CALL DLACPY( 'L', M, M, A, LDA, WORK( IU ),
$                                    LDWRKU )

```



```

      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$           WORK( IU+LDWRKU ), LDWRKU )
*
*      Generate Q in A
*      (Workspace: need 2*M*M+2*M, prefer 2*M*M+M*M*NB)
*
      CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
      IE = ITAU
      ITAUQ = IE + M
      ITAUP = ITAUQ + M
      IWORK = ITAUP + M
*
*      Bidiagonalize L in WORK(IU), copying result to
*      WORK(IR)
*      (Workspace: need 2*M*M+4*M,
*      prefer 2*M*M+3*M+2*M*NB)
*
      CALL DGEBRD( M, M, WORK( IU ), LDWRKU, S,
$           WORK( IE ), WORK( ITAUQ ),
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
      CALL DLACPY( 'L', M, M, WORK( IU ), LDWRKU,
$           WORK( IR ), LDWRKR )
*
*      Generate right bidiagonalizing vectors in WORK(IU)
*      (Workspace: need 2*M*M+4*M-1,
*      prefer 2*M*M+3*M+(M-1)*NB)
*
      CALL DORGBR( 'P', M, M, M, WORK( IU ), LDWRKU,
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
*
*      Generate left bidiagonalizing vectors in WORK(IR)
*      (Workspace: need 2*M*M+4*M, prefer 2*M*M+3*M+M*NB)
*
      CALL DORGBR( 'Q', M, M, M, WORK( IR ), LDWRKR,
$           WORK( ITAUQ ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of L in WORK(IR) and computing
*      right singular vectors of L in WORK(IU)
*      (Workspace: need 2*M*M+BDSPAC)
*
      CALL DBDSQR( 'U', M, M, M, 0, S, WORK( IE ),
$           WORK( IU ), LDWRKU, WORK( IR ),
$           LDWRKR, DUM, 1, WORK( IWORK ), INFO )
*

```

```

*          Multiply right singular vectors of L in WORK(IU) by
*          Q in A, storing result in VT
*          (Workspace: need M*M)
*
*          CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IU ),
$              LDWRKU, A, LDA, ZERO, VT, LDVT )
*
*          Copy left singular vectors of L to A
*          (Workspace: need M*M)
*
*          CALL DLACPY( 'F', M, M, WORK( IR ), LDWRKR, A,
$              LDA )
*
*          ELSE
*
*          Insufficient workspace for a fast algorithm
*
*          ITAU = 1
*          IWORK = ITAU + M
*
*          Compute A=L*Q, copying result to VT
*          (Workspace: need 2*M, prefer M+M*NB)
*
*          CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*          Generate Q in VT
*          (Workspace: need 2*M, prefer M+M*NB)
*
*          CALL DORGLQ( M, N, M, VT, LDVT, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M
*          IWORK = ITAUP + M
*
*          Zero out above L in A
*
*          CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ),
$              LDA )
*
*          Bidiagonalize L in A
*          (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*          CALL DGEBRD( M, M, A, LDA, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Multiply right vectors bidiagonalizing L by Q in VT

```

```

*           (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*           CALL DORMBR( 'P', 'L', 'T', M, N, M, A, LDA,
$               WORK( ITAUP ), VT, LDVT,
$               WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*           Generate left bidiagonalizing vectors of L in A
*           (Workspace: need 4*M, prefer 3*M+M*NB)
*
*           CALL DORGBR( 'Q', M, M, M, A, LDA, WORK( ITAUQ ),
$               WORK( IWORK ), LWORK-IWORK+1, IERR )
*           IWORK = IE + M
*
*           Perform bidiagonal QR iteration, compute left
*           singular vectors of A in A and compute right
*           singular vectors of A in VT
*           (Workspace: need BDSPAC)
*
*           CALL DBDSQR( 'U', M, N, M, 0, S, WORK( IE ), VT,
$               LDVT, A, LDA, DUM, 1, WORK( IWORK ),
$               INFO )
*
*           END IF
*
*           ELSE IF( WNTUAS ) THEN
*
*           Path 6t(N much larger than M, JOBU='S' or 'A',
*           JOBVT='S')
*           M right singular vectors to be computed in VT and
*           M left singular vectors to be computed in U
*
*           IF( LWORK.GE.M*M+MAX( 4*M, BDSPAC ) ) THEN
*
*           Sufficient workspace for a fast algorithm
*
*           IU = 1
*           IF( LWORK.GE.WRKBL+LDA*M ) THEN
*
*           WORK(IU) is LDA by N
*
*           LDWRKU = LDA
*           ELSE
*
*           WORK(IU) is LDA by M
*
*           LDWRKU = M
*           END IF
*           ITAU = IU + LDWRKU*M
*           IWORK = ITAU + M
*

```

```

*          Compute A=L*Q
*          (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*          CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Copy L to WORK(IU), zeroing out above it
*
*          CALL DLACPY( 'L', M, M, A, LDA, WORK( IU ),
$              LDWRKU )
*          CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$              WORK( IU+LDWRKU ), LDWRKU )
*
*          Generate Q in A
*          (Workspace: need M*M+2*M, prefer M*M+M+M*Nb)
*
*          CALL DORGLQ( M, N, M, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M
*          IWORK = ITAUP + M
*
*          Bidiagonalize L in WORK(IU), copying result to U
*          (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*Nb)
*
*          CALL DGEBRD( M, M, WORK( IU ), LDWRKU, S,
$              WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'L', M, M, WORK( IU ), LDWRKU, U,
$              LDU )
*
*          Generate right bidiagonalizing vectors in WORK(IU)
*          (Workspace: need M*M+4*M-1,
*              prefer M*M+3*M+(M-1)*Nb)
*
*          CALL DORGBR( 'P', M, M, M, WORK( IU ), LDWRKU,
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate left bidiagonalizing vectors in U
*          (Workspace: need M*M+4*M, prefer M*M+3*M+M*Nb)
*
*          CALL DORGBR( 'Q', M, M, M, U, LDU, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IWORK = IE + M
*
*          Perform bidiagonal QR iteration, computing left
*          singular vectors of L in U and computing right

```

```

*          singular vectors of L in WORK(IU)
*          (Workspace: need M*M+BDSPAC)
*
*          CALL DBDSQR( 'U', M, M, M, 0, S, WORK( IE ),
$              WORK( IU ), LDWRKU, U, LDU, DUM, 1,
$              WORK( IWORK ), INFO )
*
*          Multiply right singular vectors of L in WORK(IU) by
*          Q in A, storing result in VT
*          (Workspace: need M*M)
*
*          CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IU ),
$              LDWRKU, A, LDA, ZERO, VT, LDVT )
*
*          ELSE
*
*          Insufficient workspace for a fast algorithm
*
*          ITAU = 1
*          IWORK = ITAU + M
*
*          Compute A=L*Q, copying result to VT
*          (Workspace: need 2*M, prefer M+M*NB)
*
*          CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*          Generate Q in VT
*          (Workspace: need 2*M, prefer M+M*NB)
*
*          CALL DORGLQ( M, N, M, VT, LDVT, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*          Copy L to U, zeroing out above it
*
*          CALL DLACPY( 'L', M, M, A, LDA, U, LDU )
*          CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, U( 1, 2 ),
$              LDU )
*          IE = ITAU
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M
*          IWORK = ITAUP + M
*
*          Bidiagonalize L in U
*          (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*          CALL DGEBRD( M, M, U, LDU, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )

```

```

*
*      Multiply right bidiagonalizing vectors in U by Q
*      in VT
*      (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*      CALL DORMBR( 'P', 'L', 'T', M, N, M, U, LDU,
$              WORK( ITAUP ), VT, LDVT,
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Generate left bidiagonalizing vectors in U
*      (Workspace: need 4*M, prefer 3*M+M*NB)
*
*      CALL DORGBR( 'Q', M, M, M, U, LDU, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in U and computing right
*      singular vectors of A in VT
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', M, N, M, 0, S, WORK( IE ), VT,
$              LDVT, U, LDU, DUM, 1, WORK( IWORK ),
$              INFO )
*
*      END IF
*
*      END IF
*
*      ELSE IF( WNTVA ) THEN
*
*      IF( WNTUN ) THEN
*
*      Path 7t(N much larger than M, JOBU='N', JOBVT='A')
*      N right singular vectors to be computed in VT and
*      no left singular vectors to be computed
*
*      IF( LWORK.GE.M*M+MAX( N+M, 4*M, BDSPAC ) ) THEN
*
*      Sufficient workspace for a fast algorithm
*
*      IR = 1
*      IF( LWORK.GE.WRKBL+LDA*M ) THEN
*
*      WORK(IR) is LDA by M
*
*      LDWRKR = LDA
*      ELSE
*
*      WORK(IR) is M by M

```

```

*
*          LDWRKR = M
*          END IF
*          ITAU = IR + LDWRKR*M
*          IWORK = ITAU + M
*
*          Compute A=L*Q, copying result to VT
*          (Workspace: need M*M+2*M, prefer M*M+M+M*NB)
*
*          CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*          Copy L to WORK(IR), zeroing out above it
*
*          CALL DLACPY( 'L', M, M, A, LDA, WORK( IR ),
$              LDWRKR )
*          CALL DLASet( 'U', M-1, M-1, ZERO, ZERO,
$              WORK( IR+LDWRKR ), LDWRKR )
*
*          Generate Q in VT
*          (Workspace: need M*M+M+N, prefer M*M+M+N*NB)
*
*          CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*          IE = ITAU
*          ITAUQ = IE + M
*          ITAUP = ITAUQ + M
*          IWORK = ITAUP + M
*
*          Bidiagonalize L in WORK(IR)
*          (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
*          CALL DGEBRD( M, M, WORK( IR ), LDWRKR, S,
$              WORK( IE ), WORK( ITAUQ ),
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*
*          Generate right bidiagonalizing vectors in WORK(IR)
*          (Workspace: need M*M+4*M-1,
*              prefer M*M+3*M+(M-1)*NB)
*
*          CALL DORGBR( 'P', M, M, M, WORK( IR ), LDWRKR,
$              WORK( ITAUP ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
*          IWORK = IE + M
*
*          Perform bidiagonal QR iteration, computing right
*          singular vectors of L in WORK(IR)
*          (Workspace: need M*M+BDSPAC)

```

```

*
*      CALL DBDSQR( 'U', M, M, 0, 0, S, WORK( IE ),
$          WORK( IR ), LDWRKR, DUM, 1, DUM, 1,
$          WORK( IWORK ), INFO )
*
*      Multiply right singular vectors of L in WORK(IR) by
*      Q in VT, storing result in A
*      (Workspace: need M*M)
*
*      CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IR ),
$          LDWRKR, VT, LDVT, ZERO, A, LDA )
*
*      Copy right singular vectors of A from A to VT
*
*      CALL DLACPY( 'F', M, N, A, LDA, VT, LDVT )
*
ELSE
*
*      Insufficient workspace for a fast algorithm
*
*      ITAU = 1
*      IWORK = ITAU + M
*
*      Compute A=L*Q, copying result to VT
*      (Workspace: need 2*M, prefer M+M*NB)
*
*      CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*      CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*      Generate Q in VT
*      (Workspace: need M+N, prefer M+N*NB)
*
*      CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$          WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IE = ITAU
*      ITAUQ = IE + M
*      ITAUP = ITAUQ + M
*      IWORK = ITAUP + M
*
*      Zero out above L in A
*
*      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ),
$          LDA )
*
*      Bidiagonalize L in A
*      (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*      CALL DGEBRD( M, M, A, LDA, S, WORK( IE ),
$          WORK( ITAUQ ), WORK( ITAUP ),

```



```

$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*                                Multiply right bidiagonalizing vectors in A by Q
*                                in VT
*                                (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*                                CALL DORMBR( 'P', 'L', 'T', M, N, M, A, LDA,
$                                WORK( ITAUP ), VT, LDVT,
$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
                                IWORK = IE + M
*
*                                Perform bidiagonal QR iteration, computing right
*                                singular vectors of A in VT
*                                (Workspace: need BDSPAC)
*
*                                CALL DBDSQR( 'U', M, N, 0, 0, S, WORK( IE ), VT,
$                                LDVT, DUM, 1, DUM, 1, WORK( IWORK ),
$                                INFO )
*
                                END IF
*
                                ELSE IF( WNTUO ) THEN
*
*                                Path 8t(N much larger than M, JOBU='O', JOBVT='A')
*                                N right singular vectors to be computed in VT and
*                                M left singular vectors to be overwritten on A
*
*                                IF( LWORK.GE.2*M*M+MAX( N+M, 4*M, BDSPAC ) ) THEN
*
*                                    Sufficient workspace for a fast algorithm
*
*                                    IU = 1
*                                    IF( LWORK.GE.WRKBL+2*LDA*M ) THEN
*
*                                        WORK(IU) is LDA by M and WORK(IR) is LDA by M
*
*                                        LDWRKU = LDA
*                                        IR = IU + LDWRKU*M
*                                        LDWRKR = LDA
*                                    ELSE IF( LWORK.GE.WRKBL+( LDA+M )*M ) THEN
*
*                                        WORK(IU) is LDA by M and WORK(IR) is M by M
*
*                                        LDWRKU = LDA
*                                        IR = IU + LDWRKU*M
*                                        LDWRKR = M
*                                    ELSE
*
*                                        WORK(IU) is M by M and WORK(IR) is M by M

```

```

LDWRKU = M
IR = IU + LDWRKU*M
LDWRKR = M
END IF
ITAU = IR + LDWRKR*M
IWORK = ITAU + M

*
*
* Compute A=L*Q, copying result to VT
* (Workspace: need 2*M*M+2*M, prefer 2*M*M+M+M*Nb)
*
$ CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )

*
*
* Generate Q in VT
* (Workspace: need 2*M*M+M+N, prefer 2*M*M+M+N*Nb)
*
$ CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )

*
*
* Copy L to WORK(IU), zeroing out above it
*
$ CALL DLACPY( 'L', M, M, A, LDA, WORK( IU ),
$           LDWRKU )
$ CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$           WORK( IU+LDWRKU ), LDWRKU )
IE = ITAU
ITAUQ = IE + M
ITAUP = ITAUQ + M
IWORK = ITAUP + M

*
*
* Bidiagonalize L in WORK(IU), copying result to
* WORK(IR)
* (Workspace: need 2*M*M+4*M,
*           prefer 2*M*M+3*M+2*M*Nb)
*
$ CALL DGEBRD( M, M, WORK( IU ), LDWRKU, S,
$           WORK( IE ), WORK( ITAUQ ),
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )
$ CALL DLACPY( 'L', M, M, WORK( IU ), LDWRKU,
$           WORK( IR ), LDWRKR )

*
*
* Generate right bidiagonalizing vectors in WORK(IU)
* (Workspace: need 2*M*M+4*M-1,
*           prefer 2*M*M+3*M+(M-1)*Nb)
*
$ CALL DORGBR( 'P', M, M, M, WORK( IU ), LDWRKU,
$           WORK( ITAUP ), WORK( IWORK ),
$           LWORK-IWORK+1, IERR )

```

```

*
*      Generate left bidiagonalizing vectors in WORK(IR)
*      (Workspace: need 2*M*M+4*M, prefer 2*M*M+3*M+M*Nb)
*
*      CALL DORGBR( 'Q', M, M, M, WORK( IR ), LDWRKR,
$              WORK( ITAUQ ), WORK( IWORK ),
$              LWORK-IWORK+1, IERR )
      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of L in WORK(IR) and computing
*      right singular vectors of L in WORK(IU)
*      (Workspace: need 2*M*M+BDSPAC)
*
*      CALL DBDSQR( 'U', M, M, M, 0, S, WORK( IE ),
$              WORK( IU ), LDWRKU, WORK( IR ),
$              LDWRKR, DUM, 1, WORK( IWORK ), INFO )
*
*      Multiply right singular vectors of L in WORK(IU) by
*      Q in VT, storing result in A
*      (Workspace: need M*M)
*
*      CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IU ),
$              LDWRKU, VT, LDVT, ZERO, A, LDA )
*
*      Copy right singular vectors of A from A to VT
*
*      CALL DLACPY( 'F', M, N, A, LDA, VT, LDVT )
*
*      Copy left singular vectors of A from WORK(IR) to A
*
*      CALL DLACPY( 'F', M, M, WORK( IR ), LDWRKR, A,
$              LDA )
*
      ELSE
*
*      Insufficient workspace for a fast algorithm
*
*      ITAU = 1
*      IWORK = ITAU + M
*
*      Compute A=L*Q, copying result to VT
*      (Workspace: need 2*M, prefer M+M*Nb)
*
*      CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
      CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*      Generate Q in VT
*      (Workspace: need M+N, prefer M+N*Nb)

```

```

*
*      CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      IE = ITAU
*      ITAUQ = IE + M
*      ITAUP = ITAUQ + M
*      IWORK = ITAUP + M
*
*      Zero out above L in A
*
*      CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, A( 1, 2 ),
$              LDA )
*
*      Bidiagonalize L in A
*      (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
*      CALL DGEBRD( M, M, A, LDA, S, WORK( IE ),
$              WORK( ITAUQ ), WORK( ITAUP ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Multiply right bidiagonalizing vectors in A by Q
*      in VT
*      (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
*      CALL DORMBR( 'P', 'L', 'T', M, N, M, A, LDA,
$              WORK( ITAUP ), VT, LDVT,
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*      Generate left bidiagonalizing vectors in A
*      (Workspace: need 4*M, prefer 3*M+M*NB)
*
*      CALL DORGBR( 'Q', M, M, M, A, LDA, WORK( ITAUQ ),
$              WORK( IWORK ), LWORK-IWORK+1, IERR )
*      IWORK = IE + M
*
*      Perform bidiagonal QR iteration, computing left
*      singular vectors of A in A and computing right
*      singular vectors of A in VT
*      (Workspace: need BDSPAC)
*
*      CALL DBDSQR( 'U', M, N, M, 0, S, WORK( IE ), VT,
$              LDVT, A, LDA, DUM, 1, WORK( IWORK ),
$              INFO )
*
*      END IF
*
*      ELSE IF( WNTUAS ) THEN
*
*      Path 9t(N much larger than M, JOBU='S' or 'A',
*      JOBVT='A')

```

```

*          N right singular vectors to be computed in VT and
*          M left singular vectors to be computed in U
*
*          IF( LWORK.GE.M*M+MAX( N+M, 4*M, BDSPAC ) ) THEN
*
*              Sufficient workspace for a fast algorithm
*
*              IU = 1
*              IF( LWORK.GE.WRKBL+LDA*M ) THEN
*
*                  WORK(IU) is LDA by M
*
*                  LDWRKU = LDA
*              ELSE
*
*                  WORK(IU) is M by M
*
*                  LDWRKU = M
*              END IF
*              ITAU = IU + LDWRKU*M
*              IWORK = ITAU + M
*
*              Compute A=L*Q, copying result to VT
*              (Workspace: need M*M+2*M, prefer M*M+M*M*NB)
*
*              CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*              CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*
*              Generate Q in VT
*              (Workspace: need M*M+M*N, prefer M*M+M*N*NB)
*
*              CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$                  WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*              Copy L to WORK(IU), zeroing out above it
*
*              CALL DLACPY( 'L', M, M, A, LDA, WORK( IU ),
$                  LDWRKU )
*              CALL DLASET( 'U', M-1, M-1, ZERO, ZERO,
$                  WORK( IU+LDWRKU ), LDWRKU )
*
*              IE = ITAU
*              ITAUQ = IE + M
*              ITAUP = ITAUQ + M
*              IWORK = ITAUP + M
*
*              Bidiagonalize L in WORK(IU), copying result to U
*              (Workspace: need M*M+4*M, prefer M*M+3*M+2*M*NB)
*
*              CALL DGEBRD( M, M, WORK( IU ), LDWRKU, S,

```

```

$                                WORK( IE ), WORK( ITAUQ ),
$                                WORK( ITAUP ), WORK( IWORK ),
$                                LWORK-IWORK+1, IERR )
CALL DLACPY( 'L', M, M, WORK( IU ), LDWRKU, U,
$                                LDU )
*
*                                Generate right bidiagonalizing vectors in WORK(IU)
*                                (Workspace: need M*M+4*M, prefer M*M+3*M+(M-1)*NB)
*
CALL DORGEB( 'P', M, M, M, M, WORK( IU ), LDWRKU,
$                                WORK( ITAUP ), WORK( IWORK ),
$                                LWORK-IWORK+1, IERR )
*
*                                Generate left bidiagonalizing vectors in U
*                                (Workspace: need M*M+4*M, prefer M*M+3*M+M*NB)
*
CALL DORGEB( 'Q', M, M, M, M, U, LDU, WORK( ITAUQ ),
$                                WORK( IWORK ), LWORK-IWORK+1, IERR )
IWORK = IE + M
*
*                                Perform bidiagonal QR iteration, computing left
*                                singular vectors of L in U and computing right
*                                singular vectors of L in WORK(IU)
*                                (Workspace: need M*M+BDSPAC)
*
CALL DBDSQR( 'U', M, M, M, M, 0, S, WORK( IE ),
$                                WORK( IU ), LDWRKU, U, LDU, DUM, 1,
$                                WORK( IWORK ), INFO )
*
*                                Multiply right singular vectors of L in WORK(IU) by
*                                Q in VT, storing result in A
*                                (Workspace: need M*M)
*
CALL DGEMM( 'N', 'N', M, N, M, ONE, WORK( IU ),
$                                LDWRKU, VT, LDVT, ZERO, A, LDA )
*
*                                Copy right singular vectors of A from A to VT
*
CALL DLACPY( 'F', M, N, A, LDA, VT, LDVT )
*
ELSE
*
*                                Insufficient workspace for a fast algorithm
*
ITAU = 1
IWORK = ITAU + M
*
*                                Compute A=L*Q, copying result to VT
*                                (Workspace: need 2*M, prefer M*M*NB)
*

```

```

CALL DGELQF( M, N, A, LDA, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )

*
*
*   Generate Q in VT
*   (Workspace: need M+N, prefer M+N*NB)
*
CALL DORGLQ( N, N, M, VT, LDVT, WORK( ITAU ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )

*
*
*   Copy L to U, zeroing out above it
*
CALL DLACPY( 'L', M, M, A, LDA, U, LDU )
CALL DLASET( 'U', M-1, M-1, ZERO, ZERO, U( 1, 2 ),
$           LDU )

IE = ITAU
ITAUQ = IE + M
ITAUP = ITAUQ + M
IWORK = ITAUP + M

*
*
*   Bidiagonalize L in U
*   (Workspace: need 4*M, prefer 3*M+2*M*NB)
*
CALL DGEBRD( M, M, U, LDU, S, WORK( IE ),
$           WORK( ITAUQ ), WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )

*
*
*   Multiply right bidiagonalizing vectors in U by Q
*   in VT
*   (Workspace: need 3*M+N, prefer 3*M+N*NB)
*
CALL DORMBR( 'P', 'L', 'T', M, N, M, U, LDU,
$           WORK( ITAUP ), VT, LDVT,
$           WORK( IWORK ), LWORK-IWORK+1, IERR )

*
*
*   Generate left bidiagonalizing vectors in U
*   (Workspace: need 4*M, prefer 3*M+M*NB)
*
CALL DORGBR( 'Q', M, M, M, U, LDU, WORK( ITAUQ ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
IWORK = IE + M

*
*
*   Perform bidiagonal QR iteration, computing left
*   singular vectors of A in U and computing right
*   singular vectors of A in VT
*   (Workspace: need BDSPAC)
*
CALL DBDSQR( 'U', M, N, M, 0, S, WORK( IE ), VT,
$           LDVT, U, LDU, DUM, 1, WORK( IWORK ),
$           INFO )

```

```

*
*           END IF
*
*           END IF
*
*           END IF
*
*       ELSE
*
*           N .LT. MNTHR
*
*           Path 10t(N greater than M, but not much larger)
*           Reduce to bidiagonal form without LQ decomposition
*
*           IE = 1
*           ITAUQ = IE + M
*           ITAUP = ITAUQ + M
*           IWORK = ITAUP + M
*
*           Bidiagonalize A
*           (Workspace: need 3*M+N, prefer 3*M+(M+N)*NB)
*
*           CALL DGEBRD( M, N, A, LDA, S, WORK( IE ), WORK( ITAUQ ),
$                   WORK( ITAUP ), WORK( IWORK ), LWORK-IWORK+1,
$                   IERR )
*           IF( WNTUAS ) THEN
*
*               If left singular vectors desired in U, copy result to U
*               and generate left bidiagonalizing vectors in U
*               (Workspace: need 4*M-1, prefer 3*M+(M-1)*NB)
*
*               CALL DLACPY( 'L', M, M, A, LDA, U, LDU )
*               CALL DORGBR( 'Q', M, M, N, U, LDU, WORK( ITAUQ ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*           END IF
*           IF( WNTVAS ) THEN
*
*               If right singular vectors desired in VT, copy result to
*               VT and generate right bidiagonalizing vectors in VT
*               (Workspace: need 3*M+NRVT, prefer 3*M+NRVT*NB)
*
*               CALL DLACPY( 'U', M, N, A, LDA, VT, LDVT )
*               IF( WNTVA )
$                   NRVT = N
*               IF( WNTVS )
$                   NRVT = M
*               CALL DORGBR( 'P', NRVT, N, M, VT, LDVT, WORK( ITAUP ),
$                   WORK( IWORK ), LWORK-IWORK+1, IERR )
*
*           END IF
*           IF( WNTUO ) THEN

```



```

*
*       If left singular vectors desired in A, generate left
*       bidiagonalizing vectors in A
*       (Workspace: need 4*M-1, prefer 3*M+(M-1)*NB)
*
*       CALL DORGBR( 'Q', M, M, N, A, LDA, WORK( ITAUQ ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*       END IF
*       IF( WNTVO ) THEN
*
*       If right singular vectors desired in A, generate right
*       bidiagonalizing vectors in A
*       (Workspace: need 4*M, prefer 3*M+M*NB)
*
*       CALL DORGBR( 'P', M, N, M, A, LDA, WORK( ITAUP ),
$           WORK( IWORK ), LWORK-IWORK+1, IERR )
*       END IF
*       IWORK = IE + M
*       IF( WNTUAS .OR. WNTUO )
$           NRU = M
*       IF( WNTUN )
$           NRU = 0
*       IF( WNTVAS .OR. WNTVO )
$           NCVT = N
*       IF( WNTVN )
$           NCVT = 0
*       IF( ( .NOT.WNTUO ) .AND. ( .NOT.WNTVO ) ) THEN
*
*       Perform bidiagonal QR iteration, if desired, computing
*       left singular vectors in U and computing right singular
*       vectors in VT
*       (Workspace: need BDSPAC)
*
*       CALL DBDSQR( 'L', M, NCVT, NRU, 0, S, WORK( IE ), VT,
$           LDVT, U, LDU, DUM, 1, WORK( IWORK ), INFO )
*       ELSE IF( ( .NOT.WNTUO ) .AND. WNTVO ) THEN
*
*       Perform bidiagonal QR iteration, if desired, computing
*       left singular vectors in U and computing right singular
*       vectors in A
*       (Workspace: need BDSPAC)
*
*       CALL DBDSQR( 'L', M, NCVT, NRU, 0, S, WORK( IE ), A, LDA,
$           U, LDU, DUM, 1, WORK( IWORK ), INFO )
*       ELSE
*
*       Perform bidiagonal QR iteration, if desired, computing
*       left singular vectors in A and computing right singular
*       vectors in VT
*       (Workspace: need BDSPAC)

```

```

*
*          CALL DBDSQR( 'L', M, NCVT, NRU, 0, S, WORK( IE ), VT,
$          LDVT, A, LDA, DUM, 1, WORK( IWORK ), INFO )
*          END IF
*
*          END IF
*
*          END IF
*
*          If DBDSQR failed to converge, copy unconverged superdiagonals
*          to WORK( 2:MINMN )
*
*          IF( INFO.NE.0 ) THEN
*              IF( IE.GT.2 ) THEN
*                  DO 50 I = 1, MINMN - 1
*                      WORK( I+1 ) = WORK( I+IE-1 )
50              CONTINUE
*              END IF
*              IF( IE.LT.2 ) THEN
*                  DO 60 I = MINMN - 1, 1, -1
*                      WORK( I+1 ) = WORK( I+IE-1 )
60              CONTINUE
*              END IF
*          END IF
*
*          Undo scaling if necessary
*
*          IF( ISCL.EQ.1 ) THEN
*              IF( ANRM.GT.BIGNUM )
$              CALL DLASCL( 'G', 0, 0, BIGNUM, ANRM, MINMN, 1, S, MINMN,
$              IERR )
*              IF( INFO.NE.0 .AND. ANRM.GT.BIGNUM )
$              CALL DLASCL( 'G', 0, 0, BIGNUM, ANRM, MINMN-1, 1, WORK( 2 ),
$              MINMN, IERR )
*              IF( ANRM.LT.SMLNUM )
$              CALL DLASCL( 'G', 0, 0, SMLNUM, ANRM, MINMN, 1, S, MINMN,
$              IERR )
*              IF( INFO.NE.0 .AND. ANRM.LT.SMLNUM )
$              CALL DLASCL( 'G', 0, 0, SMLNUM, ANRM, MINMN-1, 1, WORK( 2 ),
$              MINMN, IERR )
*          END IF
*
*          Return optimal workspace in WORK(1)
*
*          WORK( 1 ) = MAXWRK
*
*          RETURN
*
*          End of DGESVD
*

```

END

—

— LAPACK dgesvd —

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dgesvd (jobu jobvt m n a lda s u ldu vt ldvt work lwork info)
    (declare (type (simple-array double-float (*)) work vt u s a)
              (type fixnum info lwork ldvt ldu lda n m)
              (type character jobvt jobu))
    (f2cl-lib:with-multi-array-data
      ((jobu character jobu-%data% jobu-%offset%)
       (jobvt character jobvt-%data% jobvt-%offset%)
       (a double-float a-%data% a-%offset%)
       (s double-float s-%data% s-%offset%)
       (u double-float u-%data% u-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((dum (make-array 1 :element-type 'double-float)) (anrm 0.0)
              (bignum 0.0) (eps 0.0) (smlnum 0.0) (bdspac 0) (blk 0) (chunk 0)
              (i 0) (ie 0) (ierr 0) (ir 0) (iscl 0) (itau 0) (itaup 0) (itauq 0)
              (iu 0) (iwork 0) (ldwrkr 0) (ldwrku 0) (maxwrk 0) (minmn 0)
              (minwrk 0) (mnthr 0) (ncu 0) (ncvt 0) (nru 0) (nrvt 0) (wrkbl 0)
              (lquery nil) (wntua nil) (wntuas nil) (wntun nil) (wntuo nil)
              (wntus nil) (wntva nil) (wntvas nil) (wntvn nil) (wntvo nil)
              (wntvs nil))
              (declare (type (simple-array double-float (1)) dum)
                        (type (double-float) anrm bignum eps smlnum)
                        (type fixnum bdspac blk chunk i ie ierr ir iscl
                                itau itaup itauq iu iwork ldwrkr
                                ldwrku maxwrk minmn minwrk mnthr ncu
                                ncvt nru nrvt wrkbl)
                        (type (member t nil) lquery wntua wntuas wntun wntuo wntus
                                wntva wntvas wntvn wntvo wntvs))
              (setf info 0)
              (setf minmn (min (the fixnum m) (the fixnum n)))
              (setf mnthr (ilaenv 6 "DGESVD" (f2cl-lib:f2cl-// jobu jobvt) m n 0 0))
              (setf wntua (char-equal jobu #\A))
              (setf wntus (char-equal jobu #\S))
              (setf wntuas (or wntua wntus))
              (setf wntuo (char-equal jobu #\0))
              (setf wntun (char-equal jobu #\N))
              (setf wntva (char-equal jobvt #\A))
              (setf wntvs (char-equal jobvt #\S))
              (setf wntvas (or wntva wntvs))
```

```

(setf wntvo (char-equal jobvt #\O))
(setf wntvn (char-equal jobvt #\N))
(setf minwrk 1)
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((not (or wntua wntus wntuo wntun))
   (setf info -1))
  ((or (not (or wntva wntvs wntvo wntvn)) (and wntvo wntuo))
   (setf info -2))
  ((< m 0)
   (setf info -3))
  ((< n 0)
   (setf info -4))
  ((< lda (max (the fixnum 1) (the fixnum m)))
   (setf info -6))
  ((or (< ldu 1) (and wntuas (< ldu m)))
   (setf info -9))
  ((or (< ldvt 1) (and wntva (< ldvt n)) (and wntvs (< ldvt minmn)))
   (setf info -11)))
(cond
  ((and (= info 0) (or (>= lwork 1) lquery) (> m 0) (> n 0))
   (cond
    ((>= m n)
     (setf bdspac (f2cl-lib:int-mul 5 n))
     (cond
      ((>= m mnthr)
       (cond
        (wntun
         (setf maxwrk
          (f2cl-lib:int-add n
           (f2cl-lib:int-mul n
            (ilaenv 1
             "DGEQRF" " " " m
             n -1 -1))))))
        (setf maxwrk
         (max (the fixnum maxwrk)
          (the fixnum
           (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul 2
             n
             (ilaenv 1
              "DGEBRD"
              " "
              n n
              -1
              -1)))))))
      (if (or wntvo wntvas)
       (setf maxwrk
        (max (the fixnum maxwrk)

```

```

        (the fixnum
          (f2cl-lib:int-add
            (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul
              (f2cl-lib:int-sub n 1)
              (ilaenv 1 "DORGBR" "P" n n n
                -1))))))
      (setf maxwrk
        (max (the fixnum maxwrk)
          (the fixnum bdspace)))
      (setf minwrk
        (max (the fixnum (f2cl-lib:int-mul 4 n))
          (the fixnum bdspace)))
      (setf maxwrk
        (max (the fixnum maxwrk)
          (the fixnum minwrk))))
      ((and wntuo wntvn)
        (setf wrkbl
          (f2cl-lib:int-add n
            (f2cl-lib:int-mul n
              (ilaenv 1
                "DGEQRF" " " " m
                n -1 -1))))))

      (setf wrkbl
        (max (the fixnum wrkbl)
          (the fixnum
            (f2cl-lib:int-add n
              (f2cl-lib:int-mul n
                (ilaenv
                  1
                  "DORGQR"
                  " "
                  m n
                  n
                  -1)))))))

      (setf wrkbl
        (max (the fixnum wrkbl)
          (the fixnum
            (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
              (f2cl-lib:int-mul 2
                n
                (ilaenv
                  1
                  "DGEBRD"
                  " "
                  n n
                  -1
                  -1)))))))

      (setf wrkbl
        (max (the fixnum wrkbl)

```

```

      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (f2cl-lib:int-mul n
            (ilaenv
              1
              "DORGBR"
              "Q"
              n n
              n
              -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))
(setf maxwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
        wrkbl))
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
        (f2cl-lib:int-mul m n)
        n))))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))
((and wntuo wntvas)
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
          n -1 -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGQR"
            " "
            m n
            n
            -1))))))
(setf wrkbl

```

```

(max (the fixnum wrkbl)
  (the fixnum
    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
      (f2cl-lib:int-mul 2
        n
        (ilaenv
          1
          "DGEBRD"
          " "
          n n
          -1
          -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGBR"
            "Q"
            n n
            n
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGBR" "P"
            n n n -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))
(setf maxwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
        wrkbl))
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
        (f2cl-lib:int-mul m n)
        n))))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))
(setf maxwrk

```

```

(max (the fixnum maxwrk)
    (the fixnum minwrk)))
((and wntus wntvn)
 (setf wrkbl
      (f2cl-lib:int-add n
                        (f2cl-lib:int-mul n
                                           (ilaenv 1
                                                    "DGEQRF" " " " m
                                                    n -1 -1)))))

(setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add n
                                (f2cl-lib:int-mul n
                                                     (ilaenv
                                                          1
                                                          "DORGQR"
                                                          " "
                                                          m n
                                                          n
                                                          -1)))))))

(setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                (f2cl-lib:int-mul 2
                                                    n
                                                    (ilaenv
                                                         1
                                                         "DGEBRD"
                                                         " "
                                                         n n
                                                         -1
                                                         -1)))))))

(setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
                                (f2cl-lib:int-mul n
                                                    (ilaenv
                                                         1
                                                         "DORGQR"
                                                         "Q"
                                                         n n
                                                         n
                                                         -1)))))))

(setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum bdspace)))
(setf maxwrk

```



```

        (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))
((and wntus wntvo)
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
            n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGQR"
            " "
            m n
            n
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
          (ilaenv
            1
            "DGEBRD"
            " "
            n n
            -1
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGBR"
            "Q"
            "Q"
            "Q"
            -1)))))))

```

```

n n
n
-1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGBR" "P"
            n n n -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n) wrkbl))

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))

((and wntus wntvas)
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
            n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGQR"
            " "
            m n
            n
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
          (ilaenv

```

```

1
"DGEBRD"
" "
n n
-1
-1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGBR"
            "Q"
            n n
            n
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGBR" "P"
            n n n -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))

((and wntua wntvn)
  (setf wrkbl
    (f2cl-lib:int-add n
      (f2cl-lib:int-mul n
        (ilaenv 1
          "DGEQRF" " " " m
            n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n

```

```

(f2cl-lib:int-mul m
 (ilaenv
  1
  "DORGQR"
  " "
  m m
  n
  -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
          (ilaenv
            1
            "DGEBRD"
            " "
            n n
            -1
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGBR"
            "Q"
            n n
            n
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))

((and wntua wntvo)
 (setf wrkbl
  (f2cl-lib:int-add n
    (f2cl-lib:int-mul n

```

```

                                (ilaenv 1
                                "DGEQRF" " " " m
                                n -1 -1))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add n
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGQR"
            " "
            m m
            n
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul 2
          n
          (ilaenv
            1
            "DGEBRD"
            " "
            n n
            -1
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul n
          (ilaenv
            1
            "DORGBR"
            "Q"
            n n
            n
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGBR" "P"
            n n n -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)

```

```

        (the fixnum bdspace)))
(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
        (the fixnum minwrk))))
((and wntua wntvas)
 (setf wrkbl
  (f2cl-lib:int-add n
    (f2cl-lib:int-mul n
      (ilaenv 1
        "DGEQRF" " " " m
        n -1 -1)))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add n
            (f2cl-lib:int-mul m
              (ilaenv
                1
                "DORGQR"
                " "
                m m
                n
                -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul 2
              n
              (ilaenv
                1
                "DGEBRD"
                " "
                n n
                -1
                -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul n
              (ilaenv
                1

```

```

                                "DORGBR"
                                "Q"
                                n n
                                n
                                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub n 1)
          (ilaenv 1 "DORGBR" "P"
            n n n -1)))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspac)))
(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul n n) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspac)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))))
(t
  (setf maxwrk
    (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
      (f2cl-lib:int-mul
        (f2cl-lib:int-add m n)
        (ilaenv 1 "DGEBRD" " " " m n -1 -1))))
  (if (or wntus wntuo)
    (setf maxwrk
      (max (the fixnum maxwrk)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul n
              (ilaenv
                1
                "DORGBR"
                "Q"
                m n
                n
                -1)))))))
    (if wntua
      (setf maxwrk
        (max (the fixnum maxwrk)
          (the fixnum
            (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
              (f2cl-lib:int-mul n
                (ilaenv
                  1
                  "DORGBR"
                  "Q"
                  m n
                  n
                  -1)))))))
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
            (f2cl-lib:int-mul n
              (ilaenv
                1
                "DORGBR"
                "Q"
                m n
                n
                -1)))))))
    )
  )

```

```

(f2cl-lib:int-mul m
 (ilaenv
  1
  "DORGBR"
  "Q"
  m m
  n
  -1))))))

(if (not wntvn)
  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 n)
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub n
              1)
            (ilaenv 1 "DORGBR"
              "P" n n n -1)))))))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum bdspace)))

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 n) m))
    (the fixnum bdspace)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))))

(t
  (setf bdspace (f2cl-lib:int-mul 5 m))
  (cond
    ((>= n mnthr)
     (cond
       (wntvn
        (setf maxwrk
          (f2cl-lib:int-add m
            (f2cl-lib:int-mul m
              (ilaenv 1
                "DGELQF" " " " m
                n -1 -1))))

        (setf maxwrk
          (max (the fixnum maxwrk)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                (f2cl-lib:int-mul 2
                  m
                  (ilaenv
                    1
                    "DGEBRD"

```



```

" "
m m
-1
-1))))))

(if (or wntuo wntuas)
  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum
        (f2cl-lib:int-add
          (f2cl-lib:int-mul 3 m)
          (f2cl-lib:int-mul m
            (ilaenv 1 "DORGBR"
              "Q" m m m
              -1)))))))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum bdspac)))

(setf minwrk
  (max (the fixnum (f2cl-lib:int-mul 4 m))
    (the fixnum bdspac)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))

((and wntvo wntun)
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " m
          n -1 -1))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGLQ"
            " "
            m n
            m
            -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1

```

```

                                "DGEBRD"
                                " "
                                m m
                                -1
                                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          (ilaenv 1 "DORGBR" "P"
            m m m -1)))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))
(setf maxwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
        wrkbl))
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
        (f2cl-lib:int-mul m n)
        m))))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))
((and wntvo wntuas)
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " " m
            n -1 -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGLQ"
            " "
            m n

```

```

                                m
                                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1
            "DGEBRD"
            " "
            m m
            -1
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          (ilaenv 1 "DORGBR" "P"
            m m m -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGBR"
            "Q"
            m m
            m
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
        wrkbl))
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
        (f2cl-lib:int-mul m n)
        m))))

(setf minwrk
  (max
    (the fixnum

```

```

        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
      (the fixnum bdspace)))
    (setf maxwrk
      (max (the fixnum maxwrk)
            (the fixnum minwrk))))
    ((and wntvs wntun)
     (setf wrkbl
       (f2cl-lib:int-add m
         (f2cl-lib:int-mul m
           (ilaenv 1
            "DGELQF" " " m
            n -1 -1))))))

    (setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add m
                (f2cl-lib:int-mul m
                  (ilaenv
                    1
                    "DORGLQ"
                    " "
                    m n
                    m
                    -1)))))))

    (setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                (f2cl-lib:int-mul 2
                  m
                  (ilaenv
                    1
                    "DGEBRD"
                    " "
                    m m
                    -1)))))))

    (setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum
              (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                (f2cl-lib:int-mul
                  (f2cl-lib:int-sub m 1)
                  (ilaenv 1 "DORGBR" "P"
                    m m m -1)))))))

    (setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum bdspace)))
    (setf maxwrk
      (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))

```

```

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspac)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))
((and wntvs wntuo)
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " m
          n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGLQ"
            " "
            m n
            m
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1
            "DGEBRD"
            " "
            m m
            -1
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          (ilaenv 1 "DORGBR" "P"
            m m m -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)

```

```

        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                           (f2cl-lib:int-mul m
                           (ilaenv
                             1
                             "DORGBR"
                             "Q"
                             m m
                             m
                             -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m) wrkbl))

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspace)))

(setf maxwrk
  (max (the fixnum maxwrk)
        (the fixnum minwrk)))

((and wntvs wntuas)
 (setf wrkbl
  (f2cl-lib:int-add m
                    (f2cl-lib:int-mul m
                    (ilaenv 1
                          "DGELQF" " " " m
                          n -1 -1)))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add m
                            (f2cl-lib:int-mul m
                            (ilaenv
                              1
                              "DORGLQ"
                              " " "
                              m n
                              m
                              -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                            (f2cl-lib:int-mul 2
                            m
                            (ilaenv
                              1

```

```

                                "DGEHRD"
                                " "
                                m m
                                -1
                                -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          (ilaenv 1 "DORGBR" "P"
            m m m -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGBR"
            "Q"
            m m
            m
            -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspace)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))

((and wntva wntun)
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " " " m
          n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add m
        (f2cl-lib:int-mul n

```

```

                                (ilaenv
                                  1
                                  "DORGLQ"
                                  " "
                                  n n
                                  m
                                  -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
            (ilaenv
              1
              "DGEBRD"
              " "
              m m
              -1
              -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
            (ilaenv 1 "DORGBR" "P"
              m m m -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))

(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk)))
((and wntva wntuo)
  (setf wrkbl
    (f2cl-lib:int-add m
      (f2cl-lib:int-mul m
        (ilaenv 1
          "DGELQF" " " " m
            n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)

```



```

        (the fixnum
          (f2cl-lib:int-add m
            (f2cl-lib:int-mul n
              (ilaenv
                1
                "DORGLQ"
                " "
                n n
                m
                -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul 2
          m
          (ilaenv
            1
            "DGEBRD"
            " "
            m m
            -1
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul
          (f2cl-lib:int-sub m 1)
          (ilaenv 1 "DORGBR" "P"
            m m m -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
        (f2cl-lib:int-mul m
          (ilaenv
            1
            "DORGBR"
            "Q"
            m m
            m
            -1))))))
(setf wrkbl
  (max (the fixnum wrkbl)
    (the fixnum bdspace)))
(setf maxwrk
  (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m) wrkbl))
(setf minwrk
  (max

```

```

        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
        (the fixnum bdspace)))
(setf maxwrk
  (max (the fixnum maxwrk)
        (the fixnum minwrk))))
((and wntva wntuas)
 (setf wrkbl
  (f2cl-lib:int-add m
    (f2cl-lib:int-mul m
      (ilaenv 1
        "DGELQF" " " " m
        n -1 -1))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add m
            (f2cl-lib:int-mul n
              (ilaenv
                1
                "DORGLQ"
                " "
                n n
                m
                -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
            (f2cl-lib:int-mul 2
              m
              (ilaenv
                1
                "DGEBRD"
                " "
                m m
                -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
            (f2cl-lib:int-mul
              (f2cl-lib:int-sub m 1)
              (ilaenv 1 "DORGBR" "P"
                m m m -1)))))))

(setf wrkbl
  (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)

```

```

                                (f2cl-lib:int-mul m
                                (ilaenv
                                 1
                                 "DORGBR"
                                 "Q"
                                 m m
                                 m
                                 -1))))))
    (setf wrkbl
      (max (the fixnum wrkbl)
            (the fixnum bdspace)))
    (setf maxwrk
      (f2cl-lib:int-add (f2cl-lib:int-mul m m) wrkbl))
    (setf minwrk
      (max
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
        (the fixnum bdspace)))
    (setf maxwrk
      (max (the fixnum maxwrk)
            (the fixnum minwrk))))))
  (t
   (setf maxwrk
     (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                       (f2cl-lib:int-mul
                        (f2cl-lib:int-add m n)
                        (ilaenv 1 "DGEBCD" " " m n -1 -1))))
   (if (or wntvs wntvo)
       (setf maxwrk
         (max (the fixnum maxwrk)
               (the fixnum
                 (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                   (f2cl-lib:int-mul m
                                   (ilaenv
                                    1
                                    "DORGBR"
                                    "P"
                                    m n
                                    m
                                    -1)))))))
       (if wntva
           (setf maxwrk
             (max (the fixnum maxwrk)
                   (the fixnum
                     (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
                                       (f2cl-lib:int-mul n
                                       (ilaenv
                                        1
                                        "DORGBR"
                                        "P"

```

```

n n
m
-1)))))))

(if (not wntun)
  (setf maxwrk
    (max (the fixnum maxwrk)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-mul 3 m)
          (f2cl-lib:int-mul
            (f2cl-lib:int-sub m
              1)
            (ilaenv 1 "DORGEBR"
              "Q" m m m -1)))))))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum bdspac)))

(setf minwrk
  (max
    (the fixnum
      (f2cl-lib:int-add (f2cl-lib:int-mul 3 m) n))
    (the fixnum bdspac)))

(setf maxwrk
  (max (the fixnum maxwrk)
    (the fixnum minwrk))))))

(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (the fixnum maxwrk) 'double-float)))

(cond
  ((and (< lwork minwrk) (not lquery))
    (setf info -13)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGESVD" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(cond
  ((or (= m 0) (= n 0))
    (if (>= lwork 1)
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one))
    (go end_label)))
(setf eps (dlamch "P"))
(setf smlnum (/ (f2cl-lib:fsqrt (dlamch "S")) eps))
(setf bignum (/ one smlnum))
(setf anrm (dlange "M" m n a lda dum))
(setf iscl 0)
(cond
  ((and (> anrm zero) (< anrm smlnum))
    (setf iscl 1)

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 anrm smlnum m n a lda ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf ierr var-9)))
(> anrm bignum)
(setf iscl 1)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 anrm bignum m n a lda ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf ierr var-9)))
(cond
  (>= m n)
  (cond
    (>= m mnthr)
    (cond
      (wntun
        (setf itau 1)
        (setf iwork (f2cl-lib:int-add itau n))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
          (dgeqrf m n a lda
            (f2cl-lib:array-slice work double-float (itau) ((1 *)))
            (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
            (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
          (setf ierr var-7))
        (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) zero
          zero
          (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
          lda)
        (setf ie 1)
        (setf itauq (f2cl-lib:int-add ie n))
        (setf itaup (f2cl-lib:int-add itauq n))
        (setf iwork (f2cl-lib:int-add itaup n))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
            var-9 var-10)
          (dgebrd n n a lda s
            (f2cl-lib:array-slice work double-float (ie) ((1 *)))
            (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
            (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
            (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
            (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9))
          (setf ierr var-10))

```

```

(setf ncvvt 0)
(cond
  ((or wntvo wntvas)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9)
      (dorgbr "P" n n n a lda
        (f2cl-lib:array-slice work
                               double-float
                               (itaup)
                               ((1 *)))
        (f2cl-lib:array-slice work
                               double-float
                               (iwork)
                               ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6 var-7 var-8))
      (setf ierr var-9))
    (setf ncvvt n)))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n ncvvt 0 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
    lda dum 1 dum 1
    (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                   var-7 var-8 var-9 var-10 var-11 var-12
                   var-13))
  (setf info var-14))
(if wntvas (dlacpy "F" n n a lda vt ldvt)))
((and wntuo wntvn)
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                        (max
                          (the fixnum
                            (f2cl-lib:int-mul 4 n))
                          (the fixnum bdspace))))
      (setf ir 1)
      (cond
        ((>= lwork
          (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                   (f2cl-lib:int-add

```

```

                                (f2cl-lib:int-mul lda n)
                                n)))
      (f2cl-lib:int-mul lda n)))
    (setf ldwrku lda)
    (setf ldwrkr lda))
  ((>= lwork
    (f2cl-lib:int-add
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add
            (f2cl-lib:int-mul lda n)
            n))))
      (f2cl-lib:int-mul n n)))
    (setf ldwrku lda)
    (setf ldwrkr n))
  (t
    (setf ldwrku
      (the fixnum
        (truncate (- lwork (* n n) n) n)))
    (setf ldwrkr n)))
  (setf itau
    (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
  (setf iwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6))
    (setf ierr var-7))
  (dlacpy "U" n n a lda
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr)
  (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
    zero zero
    (f2cl-lib:array-slice work
      double-float
      ((+ ir 1))
      ((1 *)))

    ldwrkr)
  (multiple-value-bind

```

```

(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
 var-8)
(dorgqr m n n a lda
 (f2cl-lib:array-slice work
                        double-float
                        (itau)
                        ((1 *)))
 (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
 (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
 ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (dgebrd n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr s
  (f2cl-lib:array-slice work double-float (ie) ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                 var-6 var-7 var-8 var-9))
 (setf ierr var-10))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
 (dorgbr "Q" n n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr

```



```

(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n 0 n 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    dum 1
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(setf iu (f2cl-lib:int-add ie n))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrku))
  (> i m) nil)
(tagbody
  (setf chunk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub m i)
          1))
      (the fixnum ldwrku)))
  (dgemm "N" "N" chunk n n one
    (f2cl-lib:array-slice a
      double-float
      (i 1)
      ((1 lda) (1 *)))

    lda
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr zero
    (f2cl-lib:array-slice work double-float (iu) ((1 *)))

```

```

ldwrku)
(dlacpy "F" chunk n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
(f2cl-lib:array-slice a
                        double-float
                        (i 1)
                        ((1 lda) (1 *)))
lda)))
(t
 (setf ie 1)
 (setf itauq (f2cl-lib:int-add ie n))
 (setf itaup (f2cl-lib:int-add itauq n))
 (setf iwork (f2cl-lib:int-add itaup n))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m n a lda s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
 (setf ierr var-10))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" m n n a lda
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n 0 m 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    dum 1 a lda dum 1
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10 var-11
                  var-12 var-13))
  (setf info var-14))))))
((and wntuo wntvas)
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                      (max
                       (the fixnum
                        (f2cl-lib:int-mul 4 n))
                       (the fixnum bdspace))))))
  (setf ir 1)
  (cond
   ((>= lwork
    (f2cl-lib:int-add
     (max (the fixnum wrkbl)
          (the fixnum
           (f2cl-lib:int-add
            (f2cl-lib:int-mul lda n)
            n))))
     (f2cl-lib:int-mul lda n))))
    (setf ldwrku lda)
    (setf ldwrkr lda))
   ((>= lwork
    (f2cl-lib:int-add
     (max (the fixnum wrkbl)
          (the fixnum
           (f2cl-lib:int-add
            (f2cl-lib:int-mul lda n)
            n))))
     (f2cl-lib:int-mul n n))))
    (setf ldwrku lda)
    (setf ldwrkr n))
  (t

```

```

(setf ldwrku
  (the fixnum
    (truncate (- lwork (* n n) n) n)))
(setf ldwrkr n)))
(setf itau
  (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda vt ldvt)
(dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
  zero zero
  (f2cl-lib:array-slice vt
    double-float
    (2 1)
    ((1 ldvt) (1 *)))
  ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorgqr m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))

```

```

(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n vt ldvt s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "L" n n vt ldvt
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" n n n
   (f2cl-lib:array-slice work double-float (ir) ((1 *)))
   ldwrkr
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n vt ldvt

```

```

(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n n 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(setf iu (f2cl-lib:int-add ie n))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i ldwrku))
  (> i m) nil)
(tagbody
  (setf chunk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub m i)
          1))
      (the fixnum ldwrku)))
  (dgemm "N" "N" chunk n n one
    (f2cl-lib:array-slice a
      double-float
      (i 1)
      ((1 lda) (1 *)))

    lda
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr zero
    (f2cl-lib:array-slice work double-float (iu) ((1 *)))

```

```

ldwrku)
(dlapcy "F" chunk n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
 (f2cl-lib:array-slice a
                        double-float
                        (i 1)
                        ((1 lda) (1 *)))
lda)))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
   (dgeqrf m n a lda
    (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
   (setf ierr var-7))
 (dlapcy "U" n n a lda vt ldvt)
 (dlaset "L" (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
  zero zero
  (f2cl-lib:array-slice vt
                        double-float
                        (2 1)
                        ((1 ldvt) (1 *)))
ldvt)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8)
 (dorgqr m n n a lda
  (f2cl-lib:array-slice work
                        double-float
                        (itau)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n vt ldvt s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n vt ldvt
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
   a lda
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8 var-9 var-10 var-11
           var-12))
(setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```



```

      var-8 var-9)
(dorgbr "P" n n n vt ldvt
(f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
(f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n m 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    vt ldvt a lda dum 1
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8 var-9 var-10 var-11
                var-12 var-13))
(setf info var-14))))))
(wntus
(cond
  (wntvn
   (cond
     ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
                        (max
                          (the fixnum
                            (f2cl-lib:int-mul 4 n))
                          (the fixnum bdspace))))
      (setf ir 1)
      (cond
        ((>= lwork
         (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
         (setf ldwrkr lda))
        (t
         (setf ldwrkr n)))
      (setf itau
        (f2cl-lib:int-add ir

```

```

                                (f2cl-lib:int-mul ldwrkr n))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ ir 1))
    ((1 *)))
  ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8)
  (dorgqr m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind

```

```

    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10)
(dgebrd n n
  (f2cl-lib:array-slice work
    double-float
    (ir)
    ((1 *)))
  ldwrkr s
  (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9)
  (dorgbr "Q" n n n
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8)))

```

```

(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n 0 n 0 s
   (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
   dum 1
   (f2cl-lib:array-slice work
    double-float
    (ir)
    ((1 *)))
   ldwrkr dum 1
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
   var-6 var-7 var-8 var-9 var-10 var-11
   var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n n one a lda
 (f2cl-lib:array-slice work double-float (ir) ((1 *)))
 ldwrkr zero u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
   (dgeqrf m n a lda
    (f2cl-lib:array-slice work
     double-float
     (itau)
     ((1 *)))
    (f2cl-lib:array-slice work
     double-float
     (iwork)
     ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
   (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
   (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```

```

      var-8)
(dorgqr m n n u ldu
  (f2cl-lib:array-slice work
    double-float
    (itau)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice a
    double-float
    (2 1)
    ((1 lda) (1 *)))
lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
  (dgebrd n n a lda s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9)))

```

```

(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n a lda
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
   u ldu
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12))
  (setf ierr var-13))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n 0 m 0 s
   (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
   dum 1 u ldu dum 1
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))))))
(wntvo
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n)
        (max
          (the fixnum
            (f2cl-lib:int-mul 4 n))
          (the fixnum bdspace))))))
  (setf iu 1)
  (cond
    ((>= lwork

```

```

        (f2cl-lib:int-add wrkbl
          (f2cl-lib:int-mul 2 lda n)))
      (setf ldwrku lda)
      (setf ir
        (f2cl-lib:int-add iu
          (f2cl-lib:int-mul ldwrku
            n)))

      (setf ldwrkr lda))
    ((>= lwork
      (f2cl-lib:int-add wrkbl
        (f2cl-lib:int-mul
          (f2cl-lib:int-add lda n)
          n)))

      (setf ldwrku lda)
      (setf ir
        (f2cl-lib:int-add iu
          (f2cl-lib:int-mul ldwrku
            n)))

      (setf ldwrkr n))
    (t
      (setf ldwrku n)
      (setf ir
        (f2cl-lib:int-add iu
          (f2cl-lib:int-mul ldwrku
            n)))

      (setf ldwrkr n)))
  (setf itau
    (f2cl-lib:int-add ir
      (f2cl-lib:int-mul ldwrkr n)))
  (setf iwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6))
    (setf ierr var-7))
  (dlacpy "U" n n a lda
    (f2cl-lib:array-slice work double-float (iu) ((1 *)))
    ldwrku)
  (dlaset "L" (f2cl-lib:int-sub n 1)

```

```

(f2cl-lib:int-sub n 1) zero zero
(f2cl-lib:array-slice work
  double-float
  ((+ iu 1))
  ((1 *)))

ldwrku)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorgqr m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))

    ldwrku s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float

```



```

                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "U" n n
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" n n n
    (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))

    ldwrku
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n
    (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))

    ldwrkr
    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)

```

```

((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n n 0 s
    (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
    (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
    ldwrku dum 1
    (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero u ldu)
(dlacpy "F" n n
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work
                             double-float
                             (itau)
                             ((1 *)))

```

```

(f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6))
(setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorgqr m n n u ldu
    (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice a
                        double-float
                        (2 1)
                        ((1 lda) (1 *)))
  lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n a lda s
    (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

```

```

(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    u ldu
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "P" n n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n m 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    a lda u ldu dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))))))
(wntvas
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul n n)
        (max
          (the fixnum
            (f2cl-lib:int-mul 4 n))
          (the fixnum bdspace))))
      (setf iu 1)
      (cond
        ((>= lwork
          (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
          (setf ldwrku lda))
        (t
          (setf ldwrku n)))
      (setf itau
        (f2cl-lib:int-add iu
          (f2cl-lib:int-mul ldwrku n)))
      (setf iwork (f2cl-lib:int-add itau n))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgeqrf m n a lda
          (f2cl-lib:array-slice work
            double-float
            (itau)
            ((1 *)))
          (f2cl-lib:array-slice work
            double-float
            (iwork)
            ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6))
(setf ierr var-7))
(dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ iu 1))
    ((1 *)))
  ldwrku)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorgqr m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)

```

```

                                ((1 *)))
(f2cl-lib:array-slice work
 double-float
 (itaup)
 ((1 *)))
(f2cl-lib:array-slice work
 double-float
 (iwork)
 ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "U" n n
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku vt ldvt)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
 (dorgbr "Q" n n n
  (f2cl-lib:array-slice work
   double-float
   (iu)
   ((1 *)))

  ldwrku
  (f2cl-lib:array-slice work
   double-float
   (itauq)
   ((1 *)))

  (f2cl-lib:array-slice work
   double-float
   (iwork)
   ((1 *)))

  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8))
(setf ierr var-9))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
 (dorgbr "P" n n n vt ldvt
  (f2cl-lib:array-slice work
   double-float
   (itaup)
   ((1 *)))

  (f2cl-lib:array-slice work
   double-float
   (iwork)

```

```

((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n n 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    vt ldvt
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n n one a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku zero u ldu))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5

```



```

                                var-6))
  (setf ierr var-7))
  (dlacpy "L" m n a lda u ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8)
    (dorgqr m n n u ldu
      (f2cl-lib:array-slice work
                            double-float
                            (itau)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (iwork)
                            ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7))
    (setf ierr var-8))
  (dlacpy "U" n n a lda vt ldvt)
  (dlaset "L" (f2cl-lib:int-sub n 1)
    (f2cl-lib:int-sub n 1) zero zero
    (f2cl-lib:array-slice vt
                          double-float
                          (2 1)
                          ((1 ldvt) (1 *)))
    ldvt)
  (setf ie itau)
  (setf itauq (f2cl-lib:int-add ie n))
  (setf itaup (f2cl-lib:int-add itauq n))
  (setf iwork (f2cl-lib:int-add itaup n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10)
    (dgebrd n n vt ldvt s
      (f2cl-lib:array-slice work
                            double-float
                            (ie)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itauq)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float
                            (itaup)
                            ((1 *)))
      (f2cl-lib:array-slice work
                            double-float

```

```

                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n vt ldvt
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    u ldu
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n vt ldvt
    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n m 0 s
    (f2cl-lib:array-slice work
                          double-float

```

```

                                (ie)
                                ((1 *)))
vt ldvt u ldu dum 1
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))

info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12 var-13))
(setf info var-14))))))
(wntua
(cond
  (wntvn
    (cond
      (>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul n n)
          (max
            (the fixnum
              (f2cl-lib:int-add n m))
            (the fixnum
              (f2cl-lib:int-mul 4 n))
            (the fixnum bdspace))))))
      (setf ir 1)
      (cond
        (>= lwork
          (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
        (setf ldwrkr lda))
      (t
        (setf ldwrkr n)))
      (setf itau
        (f2cl-lib:int-add ir
          (f2cl-lib:int-mul ldwrkr n)))
      (setf iwork (f2cl-lib:int-add itau n))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgeqrf m n a lda
          (f2cl-lib:array-slice work
            double-float
            (itau)
            ((1 *)))
          (f2cl-lib:array-slice work
            double-float
            (iwork)
            ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
          ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
          var-6)))

```

```

    (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ ir 1))
    ((1 *)))
  ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorgqr m m n u ldu
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))

```

```

(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "Q" n n n
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))

    ldwrkr
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n 0 n 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))

    dum 1
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))

    ldwrkr dum 1

```

```

(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
  info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12 var-13))
(setf info var-14))
(dgemm "N" "N" m n n one u ldu
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr zero a lda)
(dlacpy "F" m n a lda u ldu))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgeqrf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6))
    (setf ierr var-7))
  (dlacpy "L" m n a lda u ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8)
    (dorgqr m m n u ldu
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6 var-7))
    (setf ierr var-8))

```

```

(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice a
    double-float
    (2 1)
    ((1 lda) (1 *))))
lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n a lda s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n a lda
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    u ldu
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)

```

```

      ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12))
      (setf ierr var-13))
      (setf iwork (f2cl-lib:int-add ie n))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9 var-10 var-11 var-12 var-13 var-14)
        (dbdsqr "U" n 0 m 0 s
                 (f2cl-lib:array-slice work
                                         double-float
                                         (ie)
                                         ((1 *)))
                 dum 1 u ldu dum 1
                 (f2cl-lib:array-slice work
                                         double-float
                                         (iwork)
                                         ((1 *)))
                 info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9 var-10 var-11
                        var-12 var-13))
        (setf info var-14))))))
(wntvo
 (cond
  ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 n n)
                          (max
                           (the fixnum
                             (f2cl-lib:int-add n m))
                           (the fixnum
                             (f2cl-lib:int-mul 4 n))
                           (the fixnum bdspace))))))
  (setf iu 1)
  (cond
   ((>= lwork
         (f2cl-lib:int-add wrkbl
                           (f2cl-lib:int-mul 2 lda n))))
    (setf ldwrku lda)
    (setf ir
      (f2cl-lib:int-add iu
                        (f2cl-lib:int-mul ldwrku
                                           n)))
    (setf ldwrkr lda))
   ((>= lwork
         (f2cl-lib:int-add wrkbl
                           (f2cl-lib:int-mul
                            (f2cl-lib:int-add lda n)
                            n))))))

```



```

(setf ldwrku lda)
(setf ir
  (f2cl-lib:int-add iu
    (f2cl-lib:int-mul ldwrku
      n)))

(setf ldwrkr n))
(t
  (setf ldwrku n)
  (setf ir
    (f2cl-lib:int-add iu
      (f2cl-lib:int-mul ldwrku
        n)))

  (setf ldwrkr n)))
(setf itau
  (f2cl-lib:int-add ir
    (f2cl-lib:int-mul ldwrkr n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorgqr m m n u ldu
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7)))

```

```

    (setf ierr var-8))
(dlacpy "U" n n a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
  (f2cl-lib:int-sub n 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ iu 1))
    ((1 *)))
  ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd n n
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(dlacpy "U" n n
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" n n n
   (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))

   ldwrku
   (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))

   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n
   (f2cl-lib:array-slice work
                           double-float
                           (ir)
                           ((1 *)))

   ldwrkr
   (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))

   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n n 0 s
   (f2cl-lib:array-slice work

```

```

double-float
(ie)
((1 *)))
(f2cl-lib:array-slice work
double-float
(ir)
((1 *)))
ldwrkr
(f2cl-lib:array-slice work
double-float
(iu)
((1 *)))
ldwrku dum 1
(f2cl-lib:array-slice work
double-float
(iwork)
((1 *)))
info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9 var-10 var-11
var-12 var-13))
(setf info var-14))
(dgemm "N" "N" m n n one u ldu
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku zero a lda)
(dlacpy "F" m n a lda u ldu)
(dlacpy "F" n n
(f2cl-lib:array-slice work double-float (ir) ((1 *)))
ldwrkr a lda))
(t
(setf itau 1)
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
(dgeqrf m n a lda
(f2cl-lib:array-slice work
double-float
(itau)
((1 *)))
(f2cl-lib:array-slice work
double-float
(iwork)
((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6))
(setf ierr var-7))
(dlacpy "L" m n a lda u ldu)
(multiple-value-bind

```

```

        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8)
(dorgqr m m n u ldu
 (f2cl-lib:array-slice work
                        double-float
                        (itau)
                        ((1 *)))
 (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
 (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
 ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                 var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(dlaset "L" (f2cl-lib:int-sub n 1)
 (f2cl-lib:int-sub n 1) zero zero
 (f2cl-lib:array-slice a
                        double-float
                        (2 1)
                        ((1 lda) (1 *)))
 lda)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (dgebrd n n a lda s
  (f2cl-lib:array-slice work
                        double-float
                        (ie)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
  (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5

```

```

                                var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n a lda
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *))))
  u ldu
  (f2cl-lib:array-slice work
   double-float
   (iwork)
   ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12))
(setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n a lda
   (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n m 0 s
   (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
   a lda u ldu dum 1
   (f2cl-lib:array-slice work
    double-float

```

```

                                (iwork)
                                ((1 *)))

    info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8 var-9 var-10 var-11
            var-12 var-13))
(setf info var-14))))))
(wntvas
(cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul n n)
      (max
        (the fixnum
          (f2cl-lib:int-add n m))
        (the fixnum
          (f2cl-lib:int-mul 4 n))
        (the fixnum bdspace))))))
(setf iu 1)
(cond
  ((>= lwork
    (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda n)))
    (setf ldwrku lda))
  (t
    (setf ldwrku n)))
(setf itau
  (f2cl-lib:int-add iu
    (f2cl-lib:int-mul ldwrku n)))
(setf iwork (f2cl-lib:int-add itau n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlapcy "L" m n a lda u ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorgqr m m n u ldu
    (f2cl-lib:array-slice work

```

```

                                double-float
                                (itau)
                                ((1 *)))
(f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                                var-6 var-7))
(setf ierr var-8))
(dlacpy "U" n n a lda
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku)
(dlaset "L" (f2cl-lib:int-sub n 1)
(f2cl-lib:int-sub n 1) zero zero
(f2cl-lib:array-slice work
                                double-float
                                ((+ iu 1))
                                ((1 *)))

ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
(dgebrd n n
(f2cl-lib:array-slice work
                                double-float
                                (iu)
                                ((1 *)))

ldwrku s
(f2cl-lib:array-slice work
                                double-float
                                (ie)
                                ((1 *)))

(f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))

(f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))

(f2cl-lib:array-slice work
                                double-float
                                (iwork)

```



```

((1 *))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "U" n n
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" n n n
    (f2cl-lib:array-slice work
                          double-float
                          (iu)
                          ((1 *)))

    ldwrku
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n vt ldvt
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind

```

```

(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
 var-8 var-9 var-10 var-11 var-12 var-13 var-14)
(dbdsqr "U" n n n 0 s
 (f2cl-lib:array-slice work
                        double-float
                        (ie)
                        ((1 *)))

vt ldvt
(f2cl-lib:array-slice work
                        double-float
                        (iu)
                        ((1 *)))

ldwrku dum 1
(f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))

info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8 var-9 var-10 var-11
                var-12 var-13))
(setf info var-14))
(dgemm "N" "N" m n n one u ldu
 (f2cl-lib:array-slice work double-float (iu) ((1 *)))
 ldwrku zero a lda)
(dlacpy "F" m n a lda u ldu))
(t
 (setf itau 1)
 (setf iwork (f2cl-lib:int-add itau n))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgeqrf m n a lda
   (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))

   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6))
  (setf ierr var-7))
 (dlacpy "L" m n a lda u ldu)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorgqr m m n u ldu

```

```

(f2cl-lib:array-slice work
  double-float
  (itau)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7))
(setf ierr var-8))
(dlacpy "U" n n a lda vt ldvt)
(dlaset "L" (f2cl-lib:int-sub n 1)
(f2cl-lib:int-sub n 1) zero zero
(f2cl-lib:array-slice vt
  double-float
  (2 1)
  ((1 ldvt) (1 *)))
ldvt)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie n))
(setf itaup (f2cl-lib:int-add itauq n))
(setf iwork (f2cl-lib:int-add itaup n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
(dgebrd n n vt ldvt s
  (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
  (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "Q" "R" "N" m n n vt ldvt
   (f2cl-lib:array-slice work
    double-float
    (itauq)
    ((1 *)))

   u ldu
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" n n n vt ldvt
   (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))

   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie n))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n n m 0 s
   (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))

   vt ldvt u ldu dum 1
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))

```

```

        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
      (setf info var-14)))))))))
(t
  (setf ie 1)
  (setf itauq (f2cl-lib:int-add ie n))
  (setf itaup (f2cl-lib:int-add itauq n))
  (setf iwork (f2cl-lib:int-add itaup n))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10)
    (dgebrd m n a lda s
      (f2cl-lib:array-slice work double-float (ie) ((1 *)))
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9))
    (setf ierr var-10))
  (cond
    (wntuas
      (dlacpy "L" m n a lda u ldu)
      (if wntus (setf ncu n))
      (if wntua (setf ncu m))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9)
        (dorgbr "Q" m ncu n u ldu
          (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
          (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                        var-7 var-8))
        (setf ierr var-9))))
    (cond
      (wntvas
        (dlacpy "U" n n a lda vt ldvt)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9)
          (dorgbr "P" n n n vt ldvt
            (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
            (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
            (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                          var-7 var-8))
          (setf ierr var-9))))
  )

```

```

(cond
  (wntuo
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dorgbr "Q" m n n a lda
        (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8))
      (setf ierr var-9))))
  (cond
    (wntvo
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9)
        (dorgbr "P" n n n a lda
          (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
          (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
          (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8))
        (setf ierr var-9))))
    (setf iwork (f2cl-lib:int-add ie n))
    (if (or wntuas wntuo) (setf nru m))
    (if wntun (setf nru 0))
    (if (or wntvas wntvo) (setf ncvt n))
    (if wntvn (setf ncvt 0))
    (cond
      ((and (not wntuo) (not wntvo))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" n ncvt nru 0 s
            (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
            ldvt u ldu dum 1
            (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
            info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12
                           var-13))
          (setf info var-14)))
      ((and (not wntuo) wntvo)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10 var-11 var-12 var-13 var-14)
          (dbdsqr "U" n ncvt nru 0 s
            (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
            lda u ldu dum 1

```

```

        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9 var-10 var-11 var-12
                      var-13))
      (setf info var-14)))
    (t
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13 var-14)
      (dbdsqr "U" n ncvr nru 0 s
              (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
              ldvt a lda dum 1
              (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
              info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9 var-10 var-11 var-12
                      var-13))
      (setf info var-14))))))
  (t
   (cond
    ((>= n mnthr)
     (cond
      (wntvn
       (setf itau 1)
       (setf iwork (f2cl-lib:int-add itau m))
       (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgelqf m n a lda
                 (f2cl-lib:array-slice work double-float (itau) ((1 *)))
                 (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
                 (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
        (setf ierr var-7))
       (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1) zero
               zero
               (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
               lda)
       (setf ie 1)
       (setf itauq (f2cl-lib:int-add ie m))
       (setf itaup (f2cl-lib:int-add itauq m))
       (setf iwork (f2cl-lib:int-add itaup m))
       (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9 var-10)
        (dgebrd m m a lda s
                 (f2cl-lib:array-slice work double-float (ie) ((1 *)))
                 (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
                 (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
                 (f2cl-lib:array-slice work double-float (iwork) ((1 *)))

```

```

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
              var-7 var-8 var-9))
(setf ierr var-10))
(cond
  ((or wntuo wntuas)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9)
     (dorgbr "Q" m m m a lda
      (f2cl-lib:array-slice work
        double-float
        (itauq)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8))
     (setf ierr var-9))))
(setf iwork (f2cl-lib:int-add ie m))
(setf nru 0)
(if (or wntuo wntuas) (setf nru m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m 0 nru 0 s
   (f2cl-lib:array-slice work double-float (ie) ((1 *))) dum
   1 a lda dum 1
   (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9 var-10 var-11 var-12
                  var-13))
  (setf info var-14))
(if wntuas (dlacpy "F" m m a lda u ldu)))
((and wntvo wntun)
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul m m)
      (max
        (the fixnum
          (f2cl-lib:int-mul 4 m))
        (the fixnum bdspace))))
   (setf ir 1)
   (cond
    ((>= lwork

```



```

      (f2cl-lib:int-add
        (max (the fixnum wrkbl)
          (the fixnum
            (f2cl-lib:int-add
              (f2cl-lib:int-mul lda n)
              m)))
        (f2cl-lib:int-mul lda m)))
    (setf ldwrku lda)
    (setf chunk n)
    (setf ldwrkr lda))
  ((>= lwork
    (f2cl-lib:int-add
      (max (the fixnum wrkbl)
        (the fixnum
          (f2cl-lib:int-add
            (f2cl-lib:int-mul lda n)
            m)))
        (f2cl-lib:int-mul m m)))
    (setf ldwrku lda)
    (setf chunk n)
    (setf ldwrkr m))
  (t
    (setf ldwrku m)
    (setf chunk
      (the fixnum
        (truncate (- lwork (* m m) m) m)))
    (setf ldwrkr m)))
  (setf itau
    (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr m)))
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6))
    (setf ierr var-7))
  (dlacpy "L" m m a lda
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr)
  (dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)

```

```

zero zero
(f2cl-lib:array-slice work
                        double-float
                        ((+ ir ldwrkr))
                        ((1 *)))

ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq m n m a lda
   (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m
   (f2cl-lib:array-slice work double-float (ir) ((1 *)))
   ldwrkr s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))

```

```

(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m 0 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr dum 1 dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(setf iu (f2cl-lib:int-add ie m))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
  (> i n) nil)
(tagbody
  (setf blk
    (min
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub n i)
          1))
      (the fixnum chunk)))
  (dgemm "N" "N" m blk m one
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr

```

```

(f2cl-lib:array-slice a
                      double-float
                      (1 i)
                      ((1 lda) (1 *)))

lda zero
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku)
(dlacpy "F" m blk
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
(f2cl-lib:array-slice a
                      double-float
                      (1 i)
                      ((1 lda) (1 *)))

lda)))
(t
 (setf ie 1)
 (setf itauq (f2cl-lib:int-add ie m))
 (setf itaup (f2cl-lib:int-add itauq m))
 (setf iwork (f2cl-lib:int-add itaup m))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m n a lda s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9))
  (setf ierr var-10))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" m n m a lda
   (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))

```

```
(f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "L" m n 0 0 s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    a lda dum 1 dum 1
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10 var-11
                  var-12 var-13))
  (setf info var-14))))))
((and wntvo wntuas)
 (cond
  ((>= lwork
        (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                           (max
                            (the fixnum
                             (f2cl-lib:int-mul 4 m))
                            (the fixnum bdspace))))
   (setf ir 1)
   (cond
    ((>= lwork
          (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
                  (f2cl-lib:int-add
                   (f2cl-lib:int-mul lda n)
                   m)))
            (f2cl-lib:int-mul lda m)))
     (setf ldwrku lda)
     (setf chunk n)
     (setf ldwrkr lda)
     ((>= lwork
           (f2cl-lib:int-add
            (max (the fixnum wrkbl)
                 (the fixnum
```

```

                (f2cl-lib:int-add
                 (f2cl-lib:int-mul lda n)
                 m)))
        (f2cl-lib:int-mul m m)))
    (setf ldwrku lda)
    (setf chunk n)
    (setf ldwrkr m))
  (t
   (setf ldwrku m)
   (setf chunk
    (the fixnum
     (truncate (- lwork (* m m) m) m))))
  (setf ldwrkr m)))
(setf itau
 (f2cl-lib:int-add ir (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
 (dgelqf m n a lda
  (f2cl-lib:array-slice work
   double-float
   (itau)
   ((1 *)))
  (f2cl-lib:array-slice work
   double-float
   (iwork)
   ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6))
 (setf ierr var-7))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
 zero zero
 (f2cl-lib:array-slice u
  double-float
  (1 2)
  ((1 ldu) (1 *)))
ldu)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8)
 (dorglq m n m a lda
  (f2cl-lib:array-slice work
   double-float
   (itau)
   ((1 *)))
  (f2cl-lib:array-slice work
   double-float

```

```

                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
      (setf ierr var-8))
      (setf ie itau)
      (setf itauq (f2cl-lib:int-add ie m))
      (setf itaup (f2cl-lib:int-add itauq m))
      (setf iwork (f2cl-lib:int-add itaup m))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9 var-10)
        (dgebrd m m u ldu s
         (f2cl-lib:array-slice work double-float (ie) ((1 *)))
         (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *)))
         (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
         (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
         (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
         ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
        (setf ierr var-10))
      (dlacpy "U" m m u ldu
       (f2cl-lib:array-slice work double-float (ir) ((1 *)))
       ldwrkr)
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9)
        (dorgbr "P" m m m
         (f2cl-lib:array-slice work double-float (ir) ((1 *)))
         ldwrkr
         (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *)))
         (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))

```

```

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8))
(setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" m m m u ldu
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m m 0 s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (ir) ((1 *)))
   ldwrkr u ldu dum 1
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))

   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10 var-11
                  var-12 var-13))
  (setf info var-14))
(setf iu (f2cl-lib:int-add ie m))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i chunk))
  ((> i n) nil)
  (tagbody
    (setf blk
      (min
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-sub n i)
                            1))
        (the fixnum chunk))))
  (dgemm "N" "N" m blk m one

```



```

(f2cl-lib:array-slice work double-float (ir) ((1 *)))
ldwrkr
(f2cl-lib:array-slice a
                        double-float
                        (1 i)
                        ((1 lda) (1 *)))

lda zero
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
(dlacpy "F" m blk
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
(f2cl-lib:array-slice a
                        double-float
                        (1 i)
                        ((1 lda) (1 *)))

lda)))
(t
(setf itau 1)
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
   (f2cl-lib:array-slice work
                        double-float
                        (itau)
                        ((1 *)))

   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))

   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
zero zero
(f2cl-lib:array-slice u
                        double-float
                        (1 2)
                        ((1 ldu) (1 *)))

ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq m n m a lda
   (f2cl-lib:array-slice work
                        double-float

```

```

                                (itau)
                                ((1 *)))
(f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m u ldu s
    (f2cl-lib:array-slice work double-float (ie) ((1 *)))
    (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
    (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
    (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "L" "T" m n m u ldu
    (f2cl-lib:array-slice work
                        double-float
                        (itaup)
                        ((1 *)))
    a lda
    (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))

```

```

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8 var-9 var-10 var-11
            var-12))
(setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" m m m u ldu
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n m 0 s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   a lda u ldu dum 1
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10 var-11
                  var-12 var-13))
  (setf info var-14))))))
(wntvs
 (cond
  (wntun
   (cond
    (>= lwork
     (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                       (max
                        (the fixnum
                         (f2cl-lib:int-mul 4 m))
                        (the fixnum bdspace))))
    (setf ir 1)

```

```

(cond
  ((>= lwork
    (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
    (setf ldwrkr lda))
  (t
    (setf ldwrkr m)))
(setf itau
  (f2cl-lib:int-add ir
    (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "L" m m a lda
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ ir ldwrkr))
    ((1 *)))
  ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorglq m n m a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7))
(setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m
    (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))

    ldwrkr s
    (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work
                          double-float
                          (ir)
                          ((1 *)))

    ldwrkr
    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))

```

```

(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m 0 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr dum 1 dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda zero vt ldvt))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)

```

[illegible]

```

double-float
(iwork)
((1 *)))

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "L" "T" m n m a lda
   (f2cl-lib:array-slice work
    double-float
    (itaup)
    ((1 *)))
   vt ldvt
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9 var-10 var-11
var-12))
  (setf ierr var-13))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n 0 0 s
   (f2cl-lib:array-slice work
    double-float
    (ie)
    ((1 *)))
   vt ldvt dum 1 dum 1
   (f2cl-lib:array-slice work
    double-float
    (iwork)
    ((1 *)))
   info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9 var-10 var-11
var-12 var-13))
  (setf info var-14))))))
(wntuo
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m)

```



```

                                (max
                                (the fixnum
                                 (f2cl-lib:int-mul 4 m))
                                (the fixnum bdspace))))
(setf iu 1)
(cond
  ((>= lwork
    (f2cl-lib:int-add wrkbl
                      (f2cl-lib:int-mul 2 lda m)))
    (setf ldwrku lda)
    (setf ir
      (f2cl-lib:int-add iu
                        (f2cl-lib:int-mul ldwrku
                                           m)))

    (setf ldwrkr lda))
  ((>= lwork
    (f2cl-lib:int-add wrkbl
                      (f2cl-lib:int-mul
                        (f2cl-lib:int-add lda m)
                        m)))
    (setf ldwrku lda)
    (setf ir
      (f2cl-lib:int-add iu
                        (f2cl-lib:int-mul ldwrku
                                           m)))

    (setf ldwrkr m))
  (t
    (setf ldwrku m)
    (setf ir
      (f2cl-lib:int-add iu
                        (f2cl-lib:int-mul ldwrku
                                           m)))

    (setf ldwrkr m)))
(setf itau
  (f2cl-lib:int-add ir
                    (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6))
(setf ierr var-7))
(dlacpy "L" m m a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ iu ldwrku))
    ((1 *)))
  ldwrku)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq m n m a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)

```

```

                                ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "L" m m
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku
(f2cl-lib:array-slice work double-float (ir) ((1 *)))
ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
(dorgbr "P" m m m
(f2cl-lib:array-slice work
  double-float
  (iu)
  ((1 *)))

ldwrku
(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))

(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
(dorgbr "Q" m m m
(f2cl-lib:array-slice work
  double-float
  (ir)
  ((1 *)))

ldwrkr

```

```

(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku a lda zero vt ldvt)
(dlacpy "F" m m
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorglq m n m vt ldvt
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice a
    double-float
    (1 2)
    ((1 lda) (1 *)))
  lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10)
  (dgebrd m m a lda s
    (f2cl-lib:array-slice work

```

```

double-float
(ie)
((1 *)))
(f2cl-lib:array-slice work
double-float
(itaup)
((1 *)))
(f2cl-lib:array-slice work
double-float
(itaup)
((1 *)))
(f2cl-lib:array-slice work
double-float
(iwork)
((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10 var-11 var-12 var-13)
(dormbr "P" "L" "T" m n m a lda
(f2cl-lib:array-slice work
double-float
(itaup)
((1 *)))
vt ldvt
(f2cl-lib:array-slice work
double-float
(iwork)
((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9 var-10 var-11
var-12))
(setf ierr var-13))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9)
(dorgbr "Q" m m m a lda
(f2cl-lib:array-slice work
double-float
(itaup)
((1 *)))
(f2cl-lib:array-slice work
double-float
(iwork)

```

```

((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n m 0 s
    (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
    vt ldvt a lda dum 1
    (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10 var-11
                  var-12 var-13))
  (setf info var-14))))))
(wntuas
 (cond
  ((>= lwork
    (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                      (max
                       (the fixnum
                        (f2cl-lib:int-mul 4 m))
                       (the fixnum bdspace))))
   (setf iu 1)
   (cond
    ((>= lwork
      (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
     (setf ldwrku lda)
     (t
      (setf ldwrku m)))
    (setf itau
      (f2cl-lib:int-add iu
                        (f2cl-lib:int-mul ldwrku m)))
    (setf iwork (f2cl-lib:int-add itau m))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
      (dgelqf m n a lda
        (f2cl-lib:array-slice work
                               double-float
                               (itau)

```

```

((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6))
(setf ierr var-7))
(dlacpy "L" m m a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice work
    double-float
    ((+ iu ldwrku))
    ((1 *)))
  ldwrku)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8)
  (dorglq m n m a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
  (dgebrd m m
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku s

```



```

(f2cl-lib:array-slice work
  double-float
  (ie)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "L" m m
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku u ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))

    ldwrku
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "Q" m m m u ldu

```

```

(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku u ldu dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku a lda zero vt ldvt))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work

```

```

                                double-float
                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6))
(setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq m n m vt ldvt
   (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
   (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7))
(setf ierr var-8))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice u
                        double-float
                        (1 2)
                        ((1 ldu) (1 *)))
  ldu)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m u ldu s
   (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
   (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

```

```

(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "L" "T" m n m u ldu
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))

    vt ldvt
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "Q" m m m u ldu
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n m 0 s
    (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
    vt ldvt u ldu dum 1
    (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))))))
(wntva
 (cond
  (wntun
   (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                        (max
                          (the fixnum
                            (f2cl-lib:int-add n m))
                          (the fixnum
                            (f2cl-lib:int-mul 4 m))
                          (the fixnum bdspace))))
      (setf ir 1)
      (cond
       ((>= lwork
         (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
        (setf ldwrkr lda))
       (t
        (setf ldwrkr m)))
      (setf itau
        (f2cl-lib:int-add ir
                          (f2cl-lib:int-mul ldwrkr m)))
      (setf iwork (f2cl-lib:int-add itau m))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (dgelqf m n a lda
          (f2cl-lib:array-slice work
                                double-float
                                (itau)
                                ((1 *)))
          (f2cl-lib:array-slice work
                                double-float

```

```

                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6))
    (setf ierr var-7))
  (dlacpy "U" m n a lda vt ldvt)
  (dlacpy "L" m m a lda
    (f2cl-lib:array-slice work double-float (ir) ((1 *)))
    ldwrkr)
  (dlaset "U" (f2cl-lib:int-sub m 1)
    (f2cl-lib:int-sub m 1) zero zero
    (f2cl-lib:array-slice work
      double-float
      ((+ ir ldwrkr))
      ((1 *)))
    ldwrkr)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8)
    (dorglq n n m vt ldvt
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
    (setf ierr var-8))
  (setf ie itau)
  (setf itauq (f2cl-lib:int-add ie m))
  (setf itaup (f2cl-lib:int-add itauq m))
  (setf iwork (f2cl-lib:int-add itaup m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10)
    (dgebrd m m
      (f2cl-lib:array-slice work
        double-float
        (ir)
        ((1 *)))
      ldwrkr s
      (f2cl-lib:array-slice work
        double-float

```

```

                                (ie)
                                ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (itauq)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (itaup)
  ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))

    ldwrkr
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))

    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m 0 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))

```

```

(f2cl-lib:array-slice work
  double-float
  (ir)
  ((1 *)))
ldwrkr dum 1 dum 1
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))

info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12 var-13))
(setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr vt ldvt zero a lda)
(dlacpy "F" m n a lda vt ldvt))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))

      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))

      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6))
    (setf ierr var-7))
  (dlacpy "U" m n a lda vt ldvt)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8)
    (dorglq n n m vt ldvt
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))

      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))

```



```

        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7))
    (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice a
    double-float
    (1 2)
    ((1 lda) (1 *)))
lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m a lda s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
  (setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "L" "T" m n m a lda
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    vt ldvt

```

```

(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12))
(setf ierr var-13))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n 0 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    vt ldvt dum 1 dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))))))
(wntuo
  (cond
    ((>= lwork
      (f2cl-lib:int-add (f2cl-lib:int-mul 2 m m)
        (max
          (the fixnum
            (f2cl-lib:int-add n m))
          (the fixnum
            (f2cl-lib:int-mul 4 m))
          (the fixnum bdspace))))))
    (setf iu 1)
    (cond
      ((>= lwork
        (f2cl-lib:int-add wrkbl
          (f2cl-lib:int-mul 2 lda m)))
        (setf ldwrku lda)
        (setf ir
          (f2cl-lib:int-add iu
            (f2cl-lib:int-mul ldwrku
              m)))
        (setf ldwrkr lda))

```

```

(>= lwork
  (f2cl-lib:int-add wrkbl
    (f2cl-lib:int-mul
      (f2cl-lib:int-add lda m)
      m)))

(setf ldwrku lda)
(setf ir
  (f2cl-lib:int-add iu
    (f2cl-lib:int-mul ldwrku
      m)))

(setf ldwrkr m))
(t
  (setf ldwrku m)
  (setf ir
    (f2cl-lib:int-add iu
      (f2cl-lib:int-mul ldwrku
        m))))

(setf ldwrkr m)))
(setf itau
  (f2cl-lib:int-add ir
    (f2cl-lib:int-mul ldwrkr m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dgelqf m n a lda
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6))
  (setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8)
  (dorglq n n m vt ldvt
    (f2cl-lib:array-slice work
      double-float
      (itau)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)

```

```

((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7))
(setf ierr var-8))
(dlacpy "L" m m a lda
(f2cl-lib:array-slice work double-float (iu) ((1 *)))
ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
(f2cl-lib:int-sub m 1) zero zero
(f2cl-lib:array-slice work
double-float
((+ iu ldwrku))
((1 *)))

ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
(dgebrd m m
(f2cl-lib:array-slice work
double-float
(iu)
((1 *)))

ldwrku s
(f2cl-lib:array-slice work
double-float
(ie)
((1 *)))

(f2cl-lib:array-slice work
double-float
(itauq)
((1 *)))

(f2cl-lib:array-slice work
double-float
(itaup)
((1 *)))

(f2cl-lib:array-slice work
double-float
(iwork)
((1 *)))

(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8 var-9))
(setf ierr var-10))

```

```

(dlacpy "L" m m
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" m m m
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
      double-float
      (ie)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku
    (f2cl-lib:array-slice work
      double-float
      (ir)
      ((1 *)))
    ldwrkr dum 1
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8 var-9 var-10 var-11
    var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku vt ldvt zero a lda)
(dlacpy "F" m n a lda vt ldvt)
(dlacpy "F" m m
  (f2cl-lib:array-slice work double-float (ir) ((1 *)))
  ldwrkr a lda))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
        double-float
        (itau)
        ((1 *)))
      (f2cl-lib:array-slice work
        double-float
        (iwork)
        ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
            var-6))
(setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq n n m vt ldvt
    (f2cl-lib:array-slice work
                          double-float
                          (itau)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7))
  (setf ierr var-8))
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice a
                        double-float
                        (1 2)
                        ((1 lda) (1 *)))
  lda)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m a lda s
    (f2cl-lib:array-slice work
                          double-float
                          (ie)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))
    (f2cl-lib:array-slice work
                          double-float

```

```

                                (iwork)
                                ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9))
(setf ierr var-10))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13)
  (dormbr "P" "L" "T" m n m a lda
    (f2cl-lib:array-slice work
                          double-float
                          (itaup)
                          ((1 *)))

    vt ldvt
    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12))
  (setf ierr var-13))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9)
  (dorgbr "Q" m m m a lda
    (f2cl-lib:array-slice work
                          double-float
                          (itauq)
                          ((1 *)))

    (f2cl-lib:array-slice work
                          double-float
                          (iwork)
                          ((1 *)))

    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
  (setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m n m 0 s
    (f2cl-lib:array-slice work
                          double-float

```



```

                                (ie)
                                ((1 *)))
vt ldvt a lda dum 1
(f2cl-lib:array-slice work
 double-float
 (iwork)
 ((1 *)))

info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9 var-10 var-11
              var-12 var-13))
(setf info var-14))))))
(wntuas
(cond
 (>= lwork
  (f2cl-lib:int-add (f2cl-lib:int-mul m m)
                    (max
                     (the fixnum
                       (f2cl-lib:int-add n m))
                     (the fixnum
                       (f2cl-lib:int-mul 4 m))
                     (the fixnum bdspace))))))
(setf iu 1)
(cond
 (>= lwork
  (f2cl-lib:int-add wrkbl (f2cl-lib:int-mul lda m)))
 (setf ldwrku lda))
(t
 (setf ldwrku m)))
(setf itau
  (f2cl-lib:int-add iu
                    (f2cl-lib:int-mul ldwrku m)))
(setf iwork (f2cl-lib:int-add itau m))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
 (dgelqf m n a lda
  (f2cl-lib:array-slice work
   double-float
   (itau)
   ((1 *)))
  (f2cl-lib:array-slice work
   double-float
   (iwork)
   ((1 *)))
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
  ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6))
(setf ierr var-7))
(dlacpy "U" m n a lda vt ldvt)

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq n n m vt ldvt
   (f2cl-lib:array-slice work
                        double-float
                        (itau)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (iwork)
                        ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "L" m m a lda
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku)
(dlaset "U" (f2cl-lib:int-sub m 1)
  (f2cl-lib:int-sub m 1) zero zero
  (f2cl-lib:array-slice work
                        double-float
                        ((+ iu ldwrku))
                        ((1 *)))
  ldwrku)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m
   (f2cl-lib:array-slice work
                        double-float
                        (iu)
                        ((1 *)))
   ldwrku s
   (f2cl-lib:array-slice work
                        double-float
                        (ie)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float
                        (itauq)
                        ((1 *)))
   (f2cl-lib:array-slice work
                        double-float

```

```

                                (itaup)
                                ((1 *)))
(f2cl-lib:array-slice work
  double-float
  (iwork)
  ((1 *)))
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9))
(setf ierr var-10))
(dlacpy "L" m m
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku u ldu)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "P" m m m
    (f2cl-lib:array-slice work
      double-float
      (iu)
      ((1 *)))
    ldwrku
    (f2cl-lib:array-slice work
      double-float
      (itaup)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-7 var-8))
  (setf ierr var-9))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9)
  (dorgbr "Q" m m m u ldu
    (f2cl-lib:array-slice work
      double-float
      (itauq)
      ((1 *)))
    (f2cl-lib:array-slice work
      double-float
      (iwork)
      ((1 *)))
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
    ierr)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8))
(setf ierr var-9))
(setf iwork (f2cl-lib:int-add ie m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" m m m 0 s
    (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
    (f2cl-lib:array-slice work
                           double-float
                           (iu)
                           ((1 *)))
    ldwrku u ldu dum 1
    (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7 var-8 var-9 var-10 var-11
                   var-12 var-13))
  (setf info var-14))
(dgemm "N" "N" m n m one
  (f2cl-lib:array-slice work double-float (iu) ((1 *)))
  ldwrku vt ldvt zero a lda)
(dlacpy "F" m n a lda vt ldvt))
(t
  (setf itau 1)
  (setf iwork (f2cl-lib:int-add itau m))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dgelqf m n a lda
      (f2cl-lib:array-slice work
                             double-float
                             (itau)
                             ((1 *)))
      (f2cl-lib:array-slice work
                             double-float
                             (iwork)
                             ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6))
    (setf ierr var-7))
  (dlacpy "U" m n a lda vt ldvt))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8)
  (dorglq n n m vt ldvt
   (f2cl-lib:array-slice work
                           double-float
                           (itau)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
   ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                   var-6 var-7))
  (setf ierr var-8))
(dlacpy "L" m m a lda u ldu)
(dlaset "U" (f2cl-lib:int-sub m 1)
 (f2cl-lib:int-sub m 1) zero zero
 (f2cl-lib:array-slice u
                       double-float
                       (1 2)
                       ((1 ldu) (1 *))))
ldu)
(setf ie itau)
(setf itauq (f2cl-lib:int-add ie m))
(setf itaup (f2cl-lib:int-add itauq m))
(setf iwork (f2cl-lib:int-add itaup m))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (dgebrd m m u ldu s
   (f2cl-lib:array-slice work
                           double-float
                           (ie)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (itauq)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (itaup)
                           ((1 *)))
   (f2cl-lib:array-slice work
                           double-float
                           (iwork)
                           ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)

```

```

      ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
      (setf ierr var-10))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9 var-10 var-11 var-12 var-13)
        (dormbr "P" "L" "T" m n m u ldu
         (f2cl-lib:array-slice work
                                double-float
                                (itaup)
                                ((1 *))))
        vt ldvt
        (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *))))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12))
      (setf ierr var-13))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9)
        (dorgbr "Q" m m m u ldu
         (f2cl-lib:array-slice work
                                double-float
                                (itauq)
                                ((1 *))))
        (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *))))
        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1)
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8))
      (setf ierr var-9))
      (setf iwork (f2cl-lib:int-add ie m))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9 var-10 var-11 var-12 var-13 var-14)
        (dbdsqr "U" m n m 0 s
         (f2cl-lib:array-slice work
                                double-float
                                (ie)
                                ((1 *))))
        vt ldvt u ldu dum 1

```

```

        (f2cl-lib:array-slice work
                                double-float
                                (iwork)
                                ((1 *)))
        info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13))
    (setf info var-14)))))))))
(t
 (setf ie 1)
 (setf itauq (f2cl-lib:int-add ie m))
 (setf itaup (f2cl-lib:int-add itauq m))
 (setf iwork (f2cl-lib:int-add itaup m))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10)
  (dgebrd m n a lda s
   (f2cl-lib:array-slice work double-float (ie) ((1 *)))
   (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
   (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
   (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                  var-7 var-8 var-9))
  (setf ierr var-10))
 (cond
  (wntuas
   (dlacpy "L" m m a lda u ldu)
   (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9)
    (dorgbr "Q" m m n u ldu
     (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
     (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
     (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8))
    (setf ierr var-9))))
  (cond
   (wntvas
    (dlacpy "U" m n a lda vt ldvt)
    (if wntva (setf nrvt n))
    (if wntvs (setf nrvt m))
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9)
     (dorgbr "P" nrvt n m vt ldvt
      (f2cl-lib:array-slice work double-float (itaup) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))

```

```

        (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
      (setf ierr var-9))))
(cond
 (wntuo
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9)
    (dorgbr "Q" m m n a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
    (setf ierr var-9))))
 (wntvo
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9)
    (dorgbr "P" m n m a lda
      (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
      (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8))
    (setf ierr var-9))))
 (setf iwork (f2cl-lib:int-add ie m))
 (if (or wntuas wntuo) (setf nru m))
 (if wntun (setf nru 0))
 (if (or wntvas wntvo) (setf ncvt n))
 (if wntvn (setf ncvt 0))
 (cond
  ((and (not wntuo) (not wntvo))
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11 var-12 var-13 var-14)
     (dbdsqr "L" m ncvt nru 0 s
       (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
       ldvt u ldu dum 1
       (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
       info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
     (setf info var-14))))
  ((and (not wntuo) wntvo)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11 var-12 var-13 var-14)
     (dorgtr "U" m n m a lda
       (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
       (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
     (setf info var-14))))
  ((and wntuo wntvo)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11 var-12 var-13 var-14)
     (dorgtr "U" m n m a lda
       (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
       (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
     (setf info var-14))))
  (t
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11 var-12 var-13 var-14)
     (dorgtr "U" m n m a lda
       (f2cl-lib:array-slice work double-float (itauq) ((1 *)))
       (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
       (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwork) 1) ierr)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8 var-9 var-10 var-11 var-12
                       var-13))
     (setf info var-14))))
  ))

```



```

        var-9 var-10 var-11 var-12 var-13 var-14)
      (dbdsqr "L" m ncv t nru 0 s
        (f2cl-lib:array-slice work double-float (ie) ((1 *))) a
        lda u ldu dum 1
        (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7 var-8 var-9 var-10 var-11 var-12
        var-13))
      (setf info var-14)))
    (t
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9 var-10 var-11 var-12 var-13 var-14)
        (dbdsqr "L" m ncv t nru 0 s
          (f2cl-lib:array-slice work double-float (ie) ((1 *))) vt
          ldvt a lda dum 1
          (f2cl-lib:array-slice work double-float (iwork) ((1 *)))
          info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
          var-7 var-8 var-9 var-10 var-11 var-12
          var-13))
        (setf info var-14))))))
  (cond
    ((/= info 0)
      (cond
        ((> ie 2)
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i
              (f2cl-lib:int-add minmn (f2cl-lib:int-sub 1)))
              nil)
            (tagbody
              (setf (f2cl-lib:fref work-%data%
                ((f2cl-lib:int-add i 1))
                ((1 *))
                work-%offset%)
                (f2cl-lib:fref work-%data%
                  ((f2cl-lib:int-sub
                    (f2cl-lib:int-add i ie)
                    1))
                  ((1 *))
                  work-%offset%))))))
          (cond
            ((< ie 2)
              (f2cl-lib:fdo (i (f2cl-lib:int-add minmn (f2cl-lib:int-sub 1))
                (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                ((> i 1) nil)
              (tagbody
                (setf (f2cl-lib:fref work-%data%
                  ((f2cl-lib:int-add i 1))

```

```

((1 *))
work-%offset%)
(f2cl-lib:fref work-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add i ie)
    1))
  ((1 *))
  work-%offset%))))))
(cond
  (= iscl 1)
  (if (> anrm bignum)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dlascl "G" 0 0 bignum anrm minmn 1 s minmn ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                       var-7 var-8))
      (setf ierr var-9)))
    (if (and (/= info 0) (> anrm bignum))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9)
        (dlascl "G" 0 0 bignum anrm (f2cl-lib:int-sub minmn 1) 1
         (f2cl-lib:array-slice work double-float (2) ((1 *))) minmn
         ierr)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                         var-7 var-8))
        (setf ierr var-9)))
      (if (< anrm smlnum)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9)
          (dlascl "G" 0 0 smlnum anrm minmn 1 s minmn ierr)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8))
          (setf ierr var-9)))
        (if (and (/= info 0) (< anrm smlnum))
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9)
            (dlascl "G" 0 0 smlnum anrm (f2cl-lib:int-sub minmn 1) 1
             (f2cl-lib:array-slice work double-float (2) ((1 *))) minmn
             ierr)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                             var-7 var-8))
            (setf ierr var-9))))
          (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
                (coerce (the fixnum maxwrk) 'double-float)))
    end_label
  (return

```

```
(values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))
```

dgesv LAPACK

— dgesv.input —

```
)set break resume
)sys rm -f dgesv.output
)spool dgesv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dgesv.help —

```
=====
dgesv examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGESV - the solution to a real system of linear equations $A * X = B$,

SYNOPSIS

```
SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
```

```
INTEGER      INFO, LDA, LDB, N, NRHS
```

```
INTEGER      IPIV( * )
```

```
DOUBLE      PRECISION A( LDA, * ), B( LDB, * )
```

PURPOSE

DGESV computes the solution to a real system of linear equations

$A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS

matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

$$A = P * L * U,$$

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations $A * X = B$.

ARGUMENTS

- N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.
- NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the N-by-N coefficient matrix A. On exit, the factors L and U from the factorization $A = P*L*U$; the unit diagonal elements of L are not stored.
- LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.
- IPIV (output) INTEGER array, dimension (N)
The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).
- B (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if INFO = 0, the N-by-NRHS solution matrix X.
- LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.
- INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

```

      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*  -- LAPACK driver routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  March 31, 1993
*
*  .. Scalar Arguments ..
      INTEGER          INFO, LDA, LDB, N, NRHS
*
*  ..
*
*  .. Array Arguments ..
      INTEGER          IPIV( * )
      DOUBLE PRECISION A( LDA, * ), B( LDB, * )
*
*  ..
*
*  =====
*
*  .. External Subroutines ..
      EXTERNAL          DGETRF, DGETRS, XERBLA
*
*  ..
*
*  .. Intrinsic Functions ..
      INTRINSIC          MAX
*
*  ..
*
*  .. Executable Statements ..
*
*  Test the input parameters.
*
      INFO = 0
      IF( N.LT.0 ) THEN
         INFO = -1
      ELSE IF( NRHS.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      ELSE IF( LDB.LT.MAX( 1, N ) ) THEN
         INFO = -7
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGESV ', -INFO )
         RETURN
      END IF
*
*  Compute the LU factorization of A.
*
      CALL DGETRF( N, N, A, LDA, IPIV, INFO )
      IF( INFO.EQ.0 ) THEN
*
*  Solve the system A*X = B, overwriting B with X.
*
         CALL DGETRS( 'No transpose', N, NRHS, A, LDA, IPIV, B, LDB,

```

```

$          INFO )
END IF
RETURN
*
*   End of DGESV
*
END

```

— LAPACK dgesv —

```

(defun dgesv (n nrhs a lda ipiv b ldb$ info)
  (declare (type (simple-array fixnum (*)) ipiv)
            (type (simple-array double-float (*)) b a)
            (type fixnum info ldb$ lda nrhs n))
  (f2cl-lib:with-multi-array-data
    ((a double-float a-%data% a-%offset%)
     (b double-float b-%data% b-%offset%)
     (ipiv fixnum ipiv-%data% ipiv-%offset%))
    (prog ()
      (declare)
      (setf info 0)
      (cond
        ((< n 0)
         (setf info -1))
        ((< nrhs 0)
         (setf info -2))
        ((< lda (max (the fixnum 1) (the fixnum n)))
         (setf info -4))
        ((< ldb$ (max (the fixnum 1) (the fixnum n)))
         (setf info -7)))
      (cond
        ((/= info 0)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DGESV " (f2cl-lib:int-sub info))
         (go end_label)))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
        (dgetrf n n a lda ipiv info)
        (declare (ignore var-0 var-1 var-2 var-3 var-4))
        (setf info var-5))
      (cond
        ((= info 0)
         (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
          (dgetrs "No transpose" n nrhs a lda ipiv b ldb$ info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)))

```

```

        (setf info var-8))))
      (go end_label)
    end_label
    (return (values nil nil nil nil nil nil nil info))))))

```

dgetf2 LAPACK

— dgetf2.input —

```

)set break resume
)sys rm -f dgetf2.output
)spool dgetf2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgetf2.help —

```

=====
dgetf2 examples
=====

=====
Man Page Details
=====

```

NAME

DGETF2 - an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges

SYNOPSIS

SUBROUTINE DGETF2(M, N, A, LDA, IPIV, INFO)

| | |
|---------|-----------------------|
| INTEGER | INFO, LDA, M, N |
| INTEGER | IPIV(*) |
| DOUBLE | PRECISION A(LDA, *) |

PURPOSE

DGETF2 computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$).

This is the right-looking Level 2 BLAS version of the algorithm.

ARGUMENTS

- M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.
- N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the m by n matrix to be factored. On exit, the factors L and U from the factorization $A = P*L*U$; the unit diagonal elements of L are not stored.
- LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.
- IPIV (output) INTEGER array, dimension (min(M,N))
The pivot indices; for $1 \leq i \leq \min(M,N)$, row i of the matrix was interchanged with row IPIV(i).
- INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -k, the k-th argument had an illegal value
> 0: if INFO = k, U(k,k) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

— dgetf2.f —

SUBROUTINE DGETF2(M, N, A, LDA, IPIV, INFO)

*
* -- LAPACK routine (version 3.0) --


```

*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      June 30, 1992
*
*      .. Scalar Arguments ..
      INTEGER          INFO, LDA, M, N
*
*      ..
*      .. Array Arguments ..
      INTEGER          IPIV( * )
      DOUBLE PRECISION A( LDA, * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
      DOUBLE PRECISION ONE, ZERO
      PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      ..
*      .. Local Scalars ..
      INTEGER          J, JP
*
*      ..
*      .. External Functions ..
      INTEGER          IDAMAX
      EXTERNAL         IDAMAX
*
*      ..
*      .. External Subroutines ..
      EXTERNAL         DGER, DSCAL, DSWAP, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DGETF2', -INFO )
         RETURN
      END IF
*
*      Quick return if possible
*

```

```

      IF( M.EQ.0 .OR. N.EQ.0 )
$     RETURN
*
      DO 10 J = 1, MIN( M, N )
*
*       Find pivot and test for singularity.
*
      JP = J - 1 + IDAMAX( M-J+1, A( J, J ), 1 )
      IPIV( J ) = JP
      IF( A( JP, J ).NE.ZERO ) THEN
*
*       Apply the interchange to columns 1:N.
*
      IF( JP.NE.J )
$       CALL DSWAP( N, A( J, 1 ), LDA, A( JP, 1 ), LDA )
*
*       Compute elements J+1:M of J-th column.
*
      IF( J.LT.M )
$       CALL DSCAL( M-J, ONE / A( J, J ), A( J+1, J ), 1 )
*
      ELSE IF( INFO.EQ.0 ) THEN
*
      INFO = J
      END IF
*
      IF( J.LT.MIN( M, N ) ) THEN
*
*       Update trailing submatrix.
*
      CALL DGER( M-J, N-J, -ONE, A( J+1, J ), 1, A( J, J+1 ), LDA,
$       A( J+1, J+1 ), LDA )
      END IF
10 CONTINUE
      RETURN
*
*     End of DGETF2
*
      END

```

— LAPACK dgetf2 —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dgetf2 (m n a lda ipiv info)

```

```

(declare (type (simple-array fixnum (*)) ipiv)
  (type (simple-array double-float (*)) a)
  (type fixnum info lda n m))
(f2cl-lib:with-multi-array-data
  ((a double-float a-%data% a-%offset%)
   (ipiv fixnum ipiv-%data% ipiv-%offset%))
(prog ((j 0) (jp 0))
  (declare (type fixnum j jp))
  (setf info 0)
  (cond
    ((< m 0)
     (setf info -1))
    ((< n 0)
     (setf info -2))
    ((< lda (max (the fixnum 1) (the fixnum m)))
     (setf info -4)))
  (cond
    ((/= info 0)
     (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGETF2" (f2cl-lib:int-sub info))
     (go end_label)))
  (if (or (= m 0) (= n 0)) (go end_label))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j
     (min (the fixnum m)
          (the fixnum n)))
    nil)
  (tagbody
    (setf jp
      (f2cl-lib:int-add (f2cl-lib:int-sub j 1)
        (idamax
         (f2cl-lib:int-add (f2cl-lib:int-sub m j)
                           1)
         (f2cl-lib:array-slice a
                                double-float
                                (j j)
                                ((1 lda) (1 *)))
         1)))
    (setf (f2cl-lib:fref ipiv-%data% (j) ((1 *)) ipiv-%offset%) jp)
    (cond
      ((/= (f2cl-lib:fref a (jp j) ((1 lda) (1 *))) zero)
       (if (/= jp j)
        (dswap n
         (f2cl-lib:array-slice a double-float (j 1) ((1 lda) (1 *)))
         lda
         (f2cl-lib:array-slice a
                               double-float
                               (jp 1)
                               ((1 lda) (1 *))))
        (f2cl-lib:fref a (jp j) ((1 lda) (1 *)))
        (f2cl-lib:fref ipiv-%data% (j) ((1 *)) ipiv-%offset%)
        jp)
      (t
       (f2cl-lib:fref a (jp j) ((1 lda) (1 *)))
       (f2cl-lib:fref ipiv-%data% (j) ((1 *)) ipiv-%offset%)
       jp)
    )
  )

```

```

        lda))
      (if (< j m)
        (dscal (f2cl-lib:int-sub m j)
          (/ one
            (f2cl-lib:fref a-%data%
              (j j)
              ((1 lda) (1 *))
              a-%offset%))
            (f2cl-lib:array-slice a
              double-float
              ((+ j 1) j)
              ((1 lda) (1 *)))
          1)))
      ((= info 0)
        (setf info j)))
    (cond
      ((< j (min (the fixnum m) (the fixnum n)))
        (dger (f2cl-lib:int-sub m j) (f2cl-lib:int-sub n j) (- one)
          (f2cl-lib:array-slice a
            double-float
            ((+ j 1) j)
            ((1 lda) (1 *)))
          1
          (f2cl-lib:array-slice a
            double-float
            (j (f2cl-lib:int-add j 1))
            ((1 lda) (1 *)))
          lda
          (f2cl-lib:array-slice a
            double-float
            ((+ j 1) (f2cl-lib:int-add j 1))
            ((1 lda) (1 *)))
          lda))))
      (go end_label)
    end_label
    (return (values nil nil nil nil nil info))))

```

dgetrf LAPACK

— dgetrf.input —

```

)set break resume
)sys rm -f dgetrf.output
)spool dgetrf.output

```

```
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dgetrf.help —

```
=====
dgetrf examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DGETRF - an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges

SYNOPSIS

```
SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
```

```
      INTEGER      INFO, LDA, M, N
```

```
      INTEGER      IPIV( * )
```

```
      DOUBLE      PRECISION A( LDA, * )
```

PURPOSE

DGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$).

This is the right-looking Level 3 BLAS version of the algorithm.

ARGUMENTS

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the M-by-N matrix to be factored. On exit, the factors L and U from the factorization $A = P*L*U$; the unit diagonal elements of L are not stored.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

IPIV (output) INTEGER array, dimension (min(M,N))
The pivot indices; for $1 \leq i \leq \min(M,N)$, row i of the matrix was interchanged with row IPIV(i).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

— dgetrf.f —

```

SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    March 31, 1993
*
*    .. Scalar Arguments ..
      INTEGER          INFO, LDA, M, N
*
*    ..
*
*    .. Array Arguments ..
      INTEGER          IPIV( * )
      DOUBLE PRECISION A( LDA, * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
      DOUBLE PRECISION ONE
      PARAMETER        ( ONE = 1.0D+0 )
*
*    ..

```

```

*      .. Local Scalars ..
      INTEGER          I, IINFO, J, JB, NB
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DGEMM, DGETF2, DLASWP, DTRSM, XERBLA
*
*      ..
*      .. External Functions ..
      INTEGER          ILAENV
      EXTERNAL          ILAENV
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      IF( M.LT.0 ) THEN
        INFO = -1
      ELSE IF( N.LT.0 ) THEN
        INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
        INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DGETRF', -INFO )
        RETURN
      END IF
*
*      Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 )
$      RETURN
*
*      Determine the block size for this environment.
*
      NB = ILAENV( 1, 'DGETRF', ' ', M, N, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.MIN( M, N ) ) THEN
*
*        Use unblocked code.
*
        CALL DGETF2( M, N, A, LDA, IPIV, INFO )
      ELSE
*
*        Use blocked code.
*
        DO 20 J = 1, MIN( M, N ), NB
          JB = MIN( MIN( M, N )-J+1, NB )
*

```

```

*      Factor diagonal and subdiagonal blocks and test for exact
*      singularity.
*
      CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ), IINFO )
*
*      Adjust INFO and the pivot indices.
*
      IF( INFO.EQ.0 .AND. IINFO.GT.0 )
$         INFO = IINFO + J - 1
      DO 10 I = J, MIN( M, J+JB-1 )
         IPIV( I ) = J - 1 + IPIV( I )
10      CONTINUE
*
*      Apply interchanges to columns 1:J-1.
*
      CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )
*
      IF( J+JB.LE.N ) THEN
*
*         Apply interchanges to columns J+JB:N.
*
      CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
$         IPIV, 1 )
*
*      Compute block row of U.
*
      CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
$         N-J-JB+1, ONE, A( J, J ), LDA, A( J, J+JB ),
$         LDA )
      IF( J+JB.LE.M ) THEN
*
*         Update trailing submatrix.
*
      CALL DGEMM( 'No transpose', 'No transpose', M-J-JB+1,
$         N-J-JB+1, JB, -ONE, A( J+JB, J ), LDA,
$         A( J, J+JB ), LDA, ONE, A( J+JB, J+JB ),
$         LDA )
         END IF
      END IF
20      CONTINUE
      END IF
      RETURN
*
*      End of DGETRF
*
      END

```

— LAPACK dgetrf —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgetrf (m n a lda ipiv info)
    (declare (type (simple-array fixnum (*)) ipiv)
              (type (simple-array double-float (*)) a)
              (type fixnum info lda n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (ipiv fixnum ipiv-%data% ipiv-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (jb 0) (nb 0))
        (declare (type fixnum i iinfo j jb nb))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info -1))
          ((< n 0)
            (setf info -2))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info -4)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DGETRF" (f2cl-lib:int-sub info))
            (go end_label)))
          (if (or (= m 0) (= n 0)) (go end_label)))
        (setf nb (ilaenv 1 "DGETRF" " " m n -1 -1))
        (cond
          ((or (<= nb 1)
              (>= nb
                (min (the fixnum m) (the fixnum n))))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
              (dgetf2 m n a lda ipiv info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4))
              (setf info var-5)))
          (t
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j nb))
              ((> j
                (min (the fixnum m)
                    (the fixnum n)))
              nil)
            (tagbody
              (setf jb
                (min
                  (the fixnum
                    (f2cl-lib:int-add
                     (f2cl-lib:int-sub

```

```

(min (the fixnum m)
    (the fixnum n))
j)
1))
(the fixnum nb)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
  (dgetf2 (f2cl-lib:int-add (f2cl-lib:int-sub m j) 1) jb
    (f2cl-lib:array-slice a double-float (j j) ((1 lda) (1 *)))
    lda
    (f2cl-lib:array-slice ipiv fixnum (j) ((1 *)))
    iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4))
  (setf iinfo var-5))
(if (and (= info 0) (> iinfo 0))
  (setf info (f2cl-lib:int-sub (f2cl-lib:int-add iinfo j) 1)))
(f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
  (> i
    (min (the fixnum m)
        (the fixnum
          (f2cl-lib:int-add j
            jb
            (f2cl-lib:int-sub
              1))))))
  nil)
(tagbody
  (setf (f2cl-lib:fref ipiv-%data% (i) ((1 *)) ipiv-%offset%)
    (f2cl-lib:int-add (f2cl-lib:int-sub j 1)
      (f2cl-lib:fref ipiv-%data%
        (i)
        ((1 *))
        ipiv-%offset%))))))
(dlaswp (f2cl-lib:int-sub j 1) a lda j
  (f2cl-lib:int-sub (f2cl-lib:int-add j jb) 1) ipiv 1)
(cond
  (<= (f2cl-lib:int-add j jb) n)
  (dlaswp (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1)
    (f2cl-lib:array-slice a
      double-float
      (1 (f2cl-lib:int-add j jb))
      ((1 lda) (1 *)))
    lda j (f2cl-lib:int-sub (f2cl-lib:int-add j jb) 1) ipiv 1)
  (dtrsm "Left" "Lower" "No transpose" "Unit" jb
    (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1) one
    (f2cl-lib:array-slice a double-float (j j) ((1 lda) (1 *)))
    lda
    (f2cl-lib:array-slice a
      double-float
      (j (f2cl-lib:int-add j jb))
      ((1 lda) (1 *)))
    lda)
  nil)

```

```

(cond
  ((<= (f2cl-lib:int-add j jb) m)
    (dgemm "No transpose" "No transpose"
      (f2cl-lib:int-add (f2cl-lib:int-sub m j jb) 1)
      (f2cl-lib:int-add (f2cl-lib:int-sub n j jb) 1) jb (- one)
      (f2cl-lib:array-slice a
        double-float
        ((+ j jb) j)
        ((1 lda) (1 *)))
      lda
      (f2cl-lib:array-slice a
        double-float
        (j (f2cl-lib:int-add j jb))
        ((1 lda) (1 *)))
      lda one
      (f2cl-lib:array-slice a
        double-float
        ((+ j jb) (f2cl-lib:int-add j jb))
        ((1 lda) (1 *)))
      lda))))))
end_label
(return (values nil nil nil nil nil info))))

```

dgetrs LAPACK

— dgetrs.input —

```

)set break resume
)sys rm -f dgetrs.output
)spool dgetrs.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dgetrs.help —

```

=====
dgetrs examples

```

```
=====
=====
Man Page Details
=====
```

NAME

DGETRS - a system of linear equations $A * X = B$ or $A' * X = B$ with a general N-by-N matrix A using the LU factorization computed by DGETRF

SYNOPSIS

```
SUBROUTINE DGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
```

```
      CHARACTER      TRANS

      INTEGER        INFO, LDA, LDB, N, NRHS

      INTEGER        IPIV( * )

      DOUBLE         PRECISION A( LDA, * ), B( LDB, * )
```

PURPOSE

DGETRS solves a system of linear equations
 $A * X = B$ or $A' * X = B$ with a general N-by-N matrix A using the LU factorization computed by DGETRF.

ARGUMENTS

```
TRANS  (input) CHARACTER*1
        Specifies the form of the system of equations:
        = 'N':  A * X = B   (No transpose)
        = 'T':  A' * X = B  (Transpose)
        = 'C':  A' * X = B  (Conjugate transpose = Transpose)

N       (input) INTEGER
        The order of the matrix A.  N >= 0.

NRHS    (input) INTEGER
        The number of right hand sides, i.e., the number of columns of
        the matrix B.  NRHS >= 0.

A       (input) DOUBLE PRECISION array, dimension (LDA,N)
        The factors L and U from the factorization  $A = P * L * U$  as computed by DGETRF.

LDA     (input) INTEGER
        The leading dimension of the array A.  LDA >= max(1,N).

IPIV    (input) INTEGER array, dimension (N)
        The pivot indices from DGETRF; for  $1 \leq i \leq N$ , row i of the matrix
```

was interchanged with row IPIV(i).

B (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
On entry, the right hand side matrix B. On exit, the solution
matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB \geq max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

— dgetrs.f —

```

SUBROUTINE DGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  March 31, 1993
*
*  .. Scalar Arguments ..
CHARACTER          TRANS
INTEGER            INFO, LDA, LDB, N, NRHS
*
*  ..
*  .. Array Arguments ..
INTEGER            IPIV( * )
DOUBLE PRECISION   A( LDA, * ), B( LDB, * )
*
*  ..
*
=====
*
*  .. Parameters ..
DOUBLE PRECISION   ONE
PARAMETER          ( ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
LOGICAL            NOTRAN
*
*  ..
*  .. External Functions ..
LOGICAL            LSAME
EXTERNAL           LSAME
*
*  ..
*  .. External Subroutines ..
EXTERNAL           DLASWP, DTRSM, XERBLA

```

```

*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC          MAX
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
*      INFO = 0
*      NOTRAN = LSAME( TRANS, 'N' )
*      IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) .AND. .NOT.
$      LSAME( TRANS, 'C' ) ) THEN
*          INFO = -1
*      ELSE IF( N.LT.0 ) THEN
*          INFO = -2
*      ELSE IF( NRHS.LT.0 ) THEN
*          INFO = -3
*      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
*          INFO = -5
*      ELSE IF( LDB.LT.MAX( 1, N ) ) THEN
*          INFO = -8
*      END IF
*      IF( INFO.NE.0 ) THEN
*          CALL XERBLA( 'DGETRS', -INFO )
*          RETURN
*      END IF
*
*      Quick return if possible
*
*      IF( N.EQ.0 .OR. NRHS.EQ.0 )
$      RETURN
*
*      IF( NOTRAN ) THEN
*
*          Solve  $A * X = B$ .
*
*          Apply row interchanges to the right hand sides.
*
*          CALL DLASWP( NRHS, B, LDB, 1, N, IPIV, 1 )
*
*          Solve  $L * X = B$ , overwriting B with X.
*
*          CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit', N, NRHS,
$          ONE, A, LDA, B, LDB )
*
*          Solve  $U * X = B$ , overwriting B with X.
*
*          CALL DTRSM( 'Left', 'Upper', 'No transpose', 'Non-unit', N,
$          NRHS, ONE, A, LDA, B, LDB )
*      ELSE

```

```

*
*      Solve A' * X = B.
*
*      Solve U'*X = B, overwriting B with X.
*
*      CALL DTRSM( 'Left', 'Upper', 'Transpose', 'Non-unit', N, NRHS,
$              ONE, A, LDA, B, LDB )
*
*      Solve L'*X = B, overwriting B with X.
*
*      CALL DTRSM( 'Left', 'Lower', 'Transpose', 'Unit', N, NRHS, ONE,
$              A, LDA, B, LDB )
*
*      Apply row interchanges to the solution vectors.
*
*      CALL DLASWP( NRHS, B, LDB, 1, N, IPIV, -1 )
END IF
*
*      RETURN
*
*      End of DGETRS
*
*      END

```

— LAPACK dgetrs —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dgetrs (trans n nrhs a lda ipiv b ldb$ info)
    (declare (type (simple-array fixnum (*)) ipiv)
              (type (simple-array double-float (*)) b a)
              (type fixnum info ldb$ lda nrhs n)
              (type character trans))
    (f2cl-lib:with-multi-array-data
      ((trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (b double-float b-%data% b-%offset%)
       (ipiv fixnum ipiv-%data% ipiv-%offset%))
      (prog ((notran nil))
        (declare (type (member t nil) notran))
        (setf info 0)
        (setf notran (char-equal trans #\N))
        (cond
          ((and (not notran) (not (char-equal trans #\T))
                (not (char-equal trans #\C)))
           (setf info -1))

```

```

      (< n 0)
      (setf info -2))
    (< nrhs 0)
    (setf info -3))
    (< lda (max (the fixnum 1) (the fixnum n)))
    (setf info -5))
    (< ldb$ (max (the fixnum 1) (the fixnum n)))
    (setf info -8)))
  (cond
    ((/= info 0)
     (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DGETRS" (f2cl-lib:int-sub info))
     (go end_label)))
  (if (or (= n 0) (= nrhs 0)) (go end_label))
  (cond
    (notran
     (dlaswp nrhs b ldb$ 1 n ipiv 1)
     (dtrsm "Left" "Lower" "No transpose" "Unit" n nrhs one a lda b ldb$)
     (dtrsm "Left" "Upper" "No transpose" "Non-unit" n nrhs one a lda b
      ldb$))
    (t
     (dtrsm "Left" "Upper" "Transpose" "Non-unit" n nrhs one a lda b ldb$)
     (dtrsm "Left" "Lower" "Transpose" "Unit" n nrhs one a lda b ldb$)
     (dlaswp nrhs b ldb$ 1 n ipiv -1)))
  (go end_label)
end_label
(return (values nil nil nil nil nil nil nil nil info))))))

```

dhseqr LAPACK

— dhseqr.input —

```

)set break resume
)sys rm -f dhseqr.output
)spool dhseqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```


— dhseqr.help —

```
=====
dhseqr examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DHSEQR - compute the eigenvalues of a Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z T Z^{**T}$, where T is an upper quasi-triangular matrix (the Schur form), and Z is the orthogonal matrix of Schur vectors

Optionally Z may be postmultiplied into an input orthogonal matrix Q , so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the orthogonal matrix Q : $A = Q*H*Q^{**T} = (QZ)*T*(QZ)^{**T}$.

SYNOPSIS

```
SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH,  WR,  WI,  Z,  LDZ,
                  WORK, LWORK, INFO )
```

```
      INTEGER      IHI, ILO, INFO, LDH, LDZ, LWORK, N
```

```
      CHARACTER    COMPZ, JOB
```

```
      DOUBLE       PRECISION  H( LDH, * ), WI( * ), WORK( * ), WR( * ),
                        Z( LDZ, * )
```

PURPOSE

DHSEQR computes the eigenvalues of a Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z T Z^{**T}$, where T is an upper quasi-triangular matrix (the Schur form), and Z is the orthogonal matrix of Schur vectors.

Optionally Z may be postmultiplied into an input orthogonal matrix Q so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the orthogonal matrix Q : $A = Q*H*Q^{**T} = (QZ)*T*(QZ)^{**T}$.

ARGUMENTS

```
      JOB          (input) CHARACTER*1
                  = 'E': compute eigenvalues only;
                  = 'S': compute eigenvalues and the Schur form T.
```

COMPZ (input) CHARACTER*1
 = 'N': no Schur vectors are computed;
 = 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned;
 = 'V': Z must contain an orthogonal matrix Q on entry, and the product $Q*Z$ is returned.

N (input) INTEGER
 The order of the matrix H. $N \geq 0$.

ILO (input) INTEGER
 IHI (input) INTEGER
 It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL, and then passed to SGEHRD when the matrix output by DGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively.
 $1 \leq ILO \leq IHI \leq N$, if $N > 0$; $ILO=1$ and $IHI=0$, if $N=0$.

H (input/output) DOUBLE PRECISION array, dimension (LDH,N)
 On entry, the upper Hessenberg matrix H.
 On exit, if JOB = 'S', H contains the upper quasi-triangular matrix T from the Schur decomposition (the Schur form); 2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with $H(i,i) = H(i+1,i+1)$ and $H(i+1,i)*H(i,i+1) < 0$. If JOB = 'E', the contents of H are unspecified on exit.

LDH (input) INTEGER
 The leading dimension of the array H. $LDH \geq \max(1,N)$.

WR (output) DOUBLE PRECISION array, dimension (N)
 WI (output) DOUBLE PRECISION array, dimension (N)
 The real and imaginary parts, respectively, of the computed eigenvalues. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with $WI(i) > 0$ and $WI(i+1) < 0$. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with $WR(i) = H(i,i)$ and, if $H(i:i+1,i:i+1)$ is a 2-by-2 diagonal block, $WI(i) = \sqrt{H(i+1,i)*H(i,i+1)}$ and $WI(i+1) = -WI(i)$.

Z (input/output) DOUBLE PRECISION array, dimension (LDZ,N)
 If COMPZ = 'N': Z is not referenced.
 If COMPZ = 'I': on entry, Z need not be set, and on exit, Z contains the orthogonal matrix Z of the Schur vectors of H.
 If COMPZ = 'V': on entry Z must contain an N-by-N matrix Q, which is assumed to be equal to the unit matrix except for

the submatrix $Z(ILO:IHI, ILO:IHI)$; on exit Z contains $Q*Z$.
 Normally Q is the orthogonal matrix generated by DORGHR after
 the call to DGEHRD which formed the Hessenberg matrix H .

LDZ (input) INTEGER
 The leading dimension of the array Z .
 LDZ $\geq \max(1, N)$ if COMPZ = 'I' or 'V'; LDZ ≥ 1 otherwise.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
 On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
 The dimension of the array WORK. LWORK $\geq \max(1, N)$.

If LWORK = -1, then a workspace query is assumed; the routine
 only calculates the optimal size of the WORK array, returns
 this value as the first entry of the WORK array, and no error
 message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value
 > 0: if INFO = i, DHSEQR failed to compute all of the
 eigenvalues in a total of $30*(IHI-ILO+1)$ iterations;
 elements 1:i-1 and i+1:n of WR and WI contain those
 eigenvalues which have been successfully computed.

— dhseqr.f —

```

SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
$                LDZ, WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    June 30, 1999
*
*    .. Scalar Arguments ..
CHARACTER          COMPZ, JOB
INTEGER            IHI, ILO, INFO, LDH, LDZ, LWORK, N
*
*    .. Array Arguments ..
DOUBLE PRECISION   H( LDH, * ), WI( * ), WORK( * ), WR( * ),
$                 Z( LDZ, * )
*
*

```

```

* =====
*
* .. Parameters ..
DOUBLE PRECISION    ZERO, ONE, TWO
PARAMETER            ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0 )
DOUBLE PRECISION    CONST
PARAMETER            ( CONST = 1.5D+0 )
INTEGER              NSMAX, LDS
PARAMETER            ( NSMAX = 15, LDS = NSMAX )
*
* ..
* .. Local Scalars ..
LOGICAL              INITZ, LQUERY, WANTT, WANTZ
INTEGER              I, I1, I2, IERR, II, ITEMP, ITN, ITS, J, K, L,
$                   MAXB, NH, NR, NS, NV
DOUBLE PRECISION    ABSW, OVFL, SMLNUM, TAU, TEMP, TST1, ULP, UNFL
*
* ..
* .. Local Arrays ..
DOUBLE PRECISION    S( LDS, NSMAX ), V( NSMAX+1 ), VV( NSMAX+1 )
*
* ..
* .. External Functions ..
LOGICAL              LSAME
INTEGER              IDAMAX, ILAENV
DOUBLE PRECISION    DLAMCH, DLANHS, DLAPY2
EXTERNAL             LSAME, IDAMAX, ILAENV, DLAMCH, DLANHS, DLAPY2
*
* ..
* .. External Subroutines ..
EXTERNAL             DCOPY, DGEMV, DLACPY, DLAHQR, DLARFG, DLARFX,
$                   DLASET, DSCAL, XERBLA
*
* ..
* .. Intrinsic Functions ..
INTRINSIC            ABS, MAX, MIN
*
* ..
* .. Executable Statements ..
*
* Decode and test the input parameters
*
*
WANTT = LSAME( JOB, 'S' )
INITZ = LSAME( COMPZ, 'I' )
WANTZ = INITZ .OR. LSAME( COMPZ, 'V' )
*
*
INFO = 0
WORK( 1 ) = MAX( 1, N )
LQUERY = ( LWORK.EQ.-1 )
IF( .NOT.LSAME( JOB, 'E' ) .AND. .NOT.WANTT ) THEN
    INFO = -1
ELSE IF( .NOT.LSAME( COMPZ, 'N' ) .AND. .NOT.WANTZ ) THEN
    INFO = -2
ELSE IF( N.LT.0 ) THEN
    INFO = -3
ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN

```

```

        INFO = -4
    ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -5
    ELSE IF( LDH.LT.MAX( 1, N ) ) THEN
        INFO = -7
    ELSE IF( LDZ.LT.1 .OR. WANTZ .AND. LDZ.LT.MAX( 1, N ) ) THEN
        INFO = -11
    ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
        INFO = -13
    END IF
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DHSEQR', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Initialize Z, if necessary
*
    IF( INITZ )
$   CALL DLASET( 'Full', N, N, ZERO, ONE, Z, LDZ )
*
*   Store the eigenvalues isolated by DGEBAL.
*
    DO 10 I = 1, ILO - 1
        WR( I ) = H( I, I )
        WI( I ) = ZERO
10 CONTINUE
    DO 20 I = IHI + 1, N
        WR( I ) = H( I, I )
        WI( I ) = ZERO
20 CONTINUE
*
*   Quick return if possible.
*
    IF( N.EQ.0 )
$   RETURN
    IF( ILO.EQ.IHI ) THEN
        WR( ILO ) = H( ILO, ILO )
        WI( ILO ) = ZERO
        RETURN
    END IF
*
*   Set rows and columns ILO to IHI to zero below the first
*   subdiagonal.
*
    DO 40 J = ILO, IHI - 2
        DO 30 I = J + 2, N
            H( I, J ) = ZERO
30 CONTINUE

```

```

40 CONTINUE
   NH = IHI - ILO + 1
*
*   Determine the order of the multi-shift QR algorithm to be used.
*
   NS = ILAENV( 4, 'DHSEQR', JOB // COMPZ, N, ILO, IHI, -1 )
   MAXB = ILAENV( 8, 'DHSEQR', JOB // COMPZ, N, ILO, IHI, -1 )
   IF( NS.LE.2 .OR. NS.GT.NH .OR. MAXB.GE.NH ) THEN
*
*       Use the standard double-shift algorithm
*
       CALL DLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILO,
$           IHI, Z, LDZ, INFO )
       RETURN
   END IF
   MAXB = MAX( 3, MAXB )
   NS = MIN( NS, MAXB, NSMAX )
*
*   Now 2 < NS <= MAXB < NH.
*
*   Set machine-dependent constants for the stopping criterion.
*   If norm(H) <= sqrt(OVFL), overflow should not occur.
*
   UNFL = DLAMCH( 'Safe minimum' )
   OVFL = ONE / UNFL
   CALL DLABAD( UNFL, OVFL )
   ULP = DLAMCH( 'Precision' )
   SMLNUM = UNFL*( NH / ULP )
*
*   I1 and I2 are the indices of the first row and last column of H
*   to which transformations must be applied. If eigenvalues only are
*   being computed, I1 and I2 are set inside the main loop.
*
   IF( WANTT ) THEN
       I1 = 1
       I2 = N
   END IF
*
*   ITN is the total number of multiple-shift QR iterations allowed.
*
   ITN = 30*NH
*
*   The main loop begins here. I is the loop index and decreases from
*   IHI to ILO in steps of at most MAXB. Each iteration of the loop
*   works with the active submatrix in rows and columns L to I.
*   Eigenvalues I+1 to IHI have already converged. Either L = ILO or
*   H(L,L-1) is negligible so that the matrix splits.
*
   I = IHI
50 CONTINUE

```

```

      L = ILO
      IF( I.LT.ILO )
$      GO TO 170
*
*      Perform multiple-shift QR iterations on rows and columns ILO to I
*      until a submatrix of order at most MAXB splits off at the bottom
*      because a subdiagonal element has become negligible.
*
      DO 150 ITS = 0, ITN
*
*      Look for a single small subdiagonal element.
*
      DO 60 K = I, L + 1, -1
          TST1 = ABS( H( K-1, K-1 ) ) + ABS( H( K, K ) )
          IF( TST1.EQ.ZERO )
$              TST1 = DLANHS( '1', I-L+1, H( L, L ), LDH, WORK )
          IF( ABS( H( K, K-1 ) ) .LE. MAX( ULP*TST1, SMLNUM ) )
$              GO TO 70
60      CONTINUE
70      CONTINUE
      L = K
      IF( L.GT.ILO ) THEN
*
*      H(L,L-1) is negligible.
*
          H( L, L-1 ) = ZERO
      END IF
*
*      Exit from loop if a submatrix of order <= MAXB has split off.
*
      IF( L.GE.I-MAXB+1 )
$          GO TO 160
*
*      Now the active submatrix is in rows and columns L to I. If
*      eigenvalues only are being computed, only the active submatrix
*      need be transformed.
*
      IF( .NOT.WANTT ) THEN
          I1 = L
          I2 = I
      END IF
*
      IF( ITS.EQ.20 .OR. ITS.EQ.30 ) THEN
*
*      Exceptional shifts.
*
          DO 80 II = I - NS + 1, I
              WR( II ) = CONST*( ABS( H( II, II-1 ) ) +
$                  ABS( H( II, II ) ) )
              WI( II ) = ZERO

```

```

80      CONTINUE
      ELSE
*
*          Use eigenvalues of trailing submatrix of order NS as shifts.
*
      CALL DLACPY( 'Full', NS, NS, H( I-NS+1, I-NS+1 ), LDH, S,
$          LDS )
      CALL DLAHQR( .FALSE., .FALSE., NS, 1, NS, S, LDS,
$          WR( I-NS+1 ), WI( I-NS+1 ), 1, NS, Z, LDZ,
$          IERR )
      IF( IERR.GT.0 ) THEN
*
*          If DLAHQR failed to compute all NS eigenvalues, use the
*          unconverged diagonal elements as the remaining shifts.
*
      DO 90 II = 1, IERR
          WR( I-NS+II ) = S( II, II )
          WI( I-NS+II ) = ZERO
90      CONTINUE
      END IF
      END IF
*
*          Form the first column of (G-w(1)) (G-w(2)) . . . (G-w(ns))
*          where G is the Hessenberg submatrix H(L:I,L:I) and w is
*          the vector of shifts (stored in WR and WI). The result is
*          stored in the local array V.
*
      V( 1 ) = ONE
      DO 100 II = 2, NS + 1
          V( II ) = ZERO
100     CONTINUE
      NV = 1
      DO 120 J = I - NS + 1, I
          IF( WI( J ).GE.ZERO ) THEN
              IF( WI( J ).EQ.ZERO ) THEN
*
*                  real shift
*
              CALL DCOPY( NV+1, V, 1, VV, 1 )
              CALL DGEMV( 'No transpose', NV+1, NV, ONE, H( L, L ),
$                  LDH, VV, 1, -WR( J ), V, 1 )
              NV = NV + 1
              ELSE IF( WI( J ).GT.ZERO ) THEN
*
*                  complex conjugate pair of shifts
*
              CALL DCOPY( NV+1, V, 1, VV, 1 )
              CALL DGEMV( 'No transpose', NV+1, NV, ONE, H( L, L ),
$                  LDH, V, 1, -TWO*WR( J ), VV, 1 )
              ITEMP = IDAMAX( NV+1, VV, 1 )

```



```

        TEMP = ONE / MAX( ABS( VV( ITEMP ) ), SMLNUM )
        CALL DSCAL( NV+1, TEMP, VV, 1 )
        ABSW = DLAPY2( WR( J ), WI( J ) )
        TEMP = ( TEMP*ABSW )*ABSW
        CALL DGEMV( 'No transpose', NV+2, NV+1, ONE,
$           H( L, L ), LDH, VV, 1, TEMP, V, 1 )
        NV = NV + 2
    END IF

*
*   Scale V(1:NV) so that max(abs(V(i))) = 1. If V is zero,
*   reset it to the unit vector.
*

    ITEMP = IDAMAX( NV, V, 1 )
    TEMP = ABS( V( ITEMP ) )
    IF( TEMP.EQ.ZERO ) THEN
        V( 1 ) = ONE
        DO 110 II = 2, NV
            V( II ) = ZERO
110        CONTINUE
    ELSE
        TEMP = MAX( TEMP, SMLNUM )
        CALL DSCAL( NV, ONE / TEMP, V, 1 )
    END IF
END IF
120 CONTINUE

*
*   Multiple-shift QR step
*

DO 140 K = L, I - 1

*
*   The first iteration of this loop determines a reflection G
*   from the vector V and applies it from left and right to H,
*   thus creating a nonzero bulge below the subdiagonal.
*
*   Each subsequent iteration determines a reflection G to
*   restore the Hessenberg form in the (K-1)th column, and thus
*   chases the bulge one step toward the bottom of the active
*   submatrix. NR is the order of G.
*

    NR = MIN( NS+1, I-K+1 )
    IF( K.GT.L )
$       CALL DCOPY( NR, H( K, K-1 ), 1, V, 1 )
        CALL DLARFG( NR, V( 1 ), V( 2 ), 1, TAU )
        IF( K.GT.L ) THEN
            H( K, K-1 ) = V( 1 )
            DO 130 II = K + 1, I
                H( II, K-1 ) = ZERO
130            CONTINUE
        END IF
        V( 1 ) = ONE

```

```

*
*      Apply G from the left to transform the rows of the matrix in
*      columns K to I2.
*
*      CALL DLARFX( 'Left', NR, I2-K+1, V, TAU, H( K, K ), LDH,
$              WORK )
*
*      Apply G from the right to transform the columns of the
*      matrix in rows I1 to min(K+NR,I).
*
*      CALL DLARFX( 'Right', MIN( K+NR, I )-I1+1, NR, V, TAU,
$              H( I1, K ), LDH, WORK )
*
*      IF( WANTZ ) THEN
*
*          Accumulate transformations in the matrix Z
*
*          CALL DLARFX( 'Right', NH, NR, V, TAU, Z( ILO, K ), LDZ,
$              WORK )
*
*      END IF
140  CONTINUE
*
150  CONTINUE
*
*      Failure to converge in remaining number of iterations
*
*      INFO = I
*      RETURN
*
160  CONTINUE
*
*      A submatrix of order <= MAXB in rows and columns L to I has split
*      off. Use the double-shift QR algorithm to handle it.
*
*      CALL DLAHQR( WANTT, WANTZ, N, L, I, H, LDH, WR, WI, ILO, IHI, Z,
$              LDZ, INFO )
*      IF( INFO.GT.0 )
$      RETURN
*
*      Decrement number of remaining iterations, and return to start of
*      the main loop with a new value of I.
*
*      ITN = ITN - ITS
*      I = L - 1
*      GO TO 50
*
170  CONTINUE
*      WORK( 1 ) = MAX( 1, N )
*      RETURN
*

```

```
*      End of DHSEQR
*
      END
```

— LAPACK dhseqr —

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (const 1.5) (nsmax 15) (lds nsmax))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 1.5 1.5) const)
            (type (fixnum 15 15) nsmax)
            (type fixnum lds))
  (defun dhseqr (job compz n ilo ihi h ldh wr wi z ldz work lwork info)
    (declare (type (simple-array double-float (*)) work z wi wr h)
              (type fixnum info lwork ldz ldh ihi ilo n)
              (type character compz job))
    (f2cl-lib:with-multi-array-data
      ((job character job-%data% job-%offset%)
       (compz character compz-%data% compz-%offset%)
       (h double-float h-%data% h-%offset%)
       (wr double-float wr-%data% wr-%offset%)
       (wi double-float wi-%data% wi-%offset%)
       (z double-float z-%data% z-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((s
               (make-array (the fixnum (reduce #'* (list lds nsmax)))
                           :element-type 'double-float))
              (v
               (make-array (f2cl-lib:int-add nsmax 1)
                           :element-type 'double-float))
              (vv
               (make-array (f2cl-lib:int-add nsmax 1)
                           :element-type 'double-float))
              (absw 0.0) (ovfl 0.0) (smlnum 0.0) (tau 0.0) (temp 0.0) (tst1 0.0)
              (ulp 0.0) (unfl 0.0) (i 0) (i1 0) (i2 0) (ierr 0) (ii 0) (itemp 0)
              (itn 0) (its 0) (j 0) (k 0) (l 0) (maxb 0) (nh 0) (nr 0) (ns 0)
              (nv 0) (initz nil) (lquery nil) (wantt nil) (wantz nil))
        (declare (type (simple-array double-float (*)) s v vv)
                  (type (double-float) absw ovfl smlnum tau temp tst1 ulp unfl)
                  (type fixnum i i1 i2 ierr ii itemp itn its j k l
                              maxb nh nr ns nv)
                  (type (member t nil) initz lquery wantt wantz))
        (setf wantt (char-equal job #\S))
        (setf initz (char-equal compz #\I))
        (setf wantz (or initz (char-equal compz #\V))))
```

```

(setf info 0)
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce
        (the fixnum
          (max (the fixnum 1)
                (the fixnum n))))
        'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((and (not (char-equal job #\E)) (not wantt))
    (setf info -1))
  ((and (not (char-equal compz #\N)) (not wantz))
    (setf info -2))
  ((< n 0)
    (setf info -3))
  ((or (< ilo 1)
        (> ilo
          (max (the fixnum 1) (the fixnum n))))
    (setf info -4))
  ((or
    (< ihi (min (the fixnum ilo) (the fixnum n)))
    (> ihi n))
    (setf info -5))
  ((< ldh (max (the fixnum 1) (the fixnum n)))
    (setf info -7))
  ((or (< ldz 1)
        (and wantz
          (< ldz
            (max (the fixnum 1)
                  (the fixnum n))))))
    (setf info -11))
  ((and
    (< lwork (max (the fixnum 1) (the fixnum n)))
    (not lquery))
    (setf info -13)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DHSEQR" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(if initz (dlaset "Full" n n zero one z ldz))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add ilo (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
          (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
    (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero)))

```

```

(f2cl-lib:fdo (i (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add i 1))
  (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
      (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
    (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero)))
(if (= n 0) (go end_label))
(cond
  ((= ilo ihi)
    (setf (f2cl-lib:fref wr-%data% (ilo) ((1 *)) wr-%offset%)
      (f2cl-lib:fref h-%data%
        (ilo ilo)
        ((1 ldh) (1 *))
        h-%offset%))
    (setf (f2cl-lib:fref wi-%data% (ilo) ((1 *)) wi-%offset%) zero)
    (go end_label)))
(f2cl-lib:fdo (j ilo (f2cl-lib:int-add j 1))
  (> j (f2cl-lib:int-add ihi (f2cl-lib:int-sub 2))) nil)
  (tagbody
    (f2cl-lib:fdo (i (f2cl-lib:int-add j 2) (f2cl-lib:int-add i 1))
      (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref h-%data% (i j) ((1 ldh) (1 *)) h-%offset%)
          zero))))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(setf ns (ilaenv 4 "DHSEQR" (f2cl-lib:f2cl-// job compz) n ilo ihi -1))
(setf maxb
  (ilaenv 8 "DHSEQR" (f2cl-lib:f2cl-// job compz) n ilo ihi -1))
(cond
  ((or (<= ns 2) (> ns nh) (>= maxb nh))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13)
      (dlahqr wantt wantz n ilo ihi h ldh wr wi ilo ihi z ldz info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12))
      (setf info var-13))
    (go end_label)))
(setf maxb
  (max (the fixnum 3) (the fixnum maxb)))
(setf ns
  (min (the fixnum ns)
    (the fixnum maxb)
    (the fixnum nsmax)))
(setf unfl (dlamch "Safe minimum"))
(setf ovfl (/ one unfl))
(multiple-value-bind (var-0 var-1)
  (dlabad unfl ovfl)
  (declare (ignore))
  (setf unfl var-0))

```

```

        (setf ovfl var-1))
      (setf ulp (dlamch "Precision"))
      (setf smlnum (* unfl (/ nh ulp)))
      (cond
        (wantt
          (setf i1 1)
          (setf i2 n)))
        (setf itn (f2cl-lib:int-mul 30 nh))
        (setf i ihi)
      label50
        (setf l ilo)
        (if (< i ilo) (go label170))
        (f2cl-lib:fdo (its 0 (f2cl-lib:int-add its 1))
          (> its itn) nil)
        (tagbody
          (f2cl-lib:fdo (k i (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
            (> k (f2cl-lib:int-add 1 1)) nil)
          (tagbody
            (setf tst1
              (+
                (abs
                  (f2cl-lib:fref h-%data%
                                ((f2cl-lib:int-sub k 1)
                                 (f2cl-lib:int-sub k 1))
                                ((1 ldh) (1 *))
                                h-%offset%))
                (abs
                  (f2cl-lib:fref h-%data%
                                (k k)
                                ((1 ldh) (1 *))
                                h-%offset%))))
              (if (= tst1 zero)
                (setf tst1
                  (dlanhs "1"
                    (f2cl-lib:int-add (f2cl-lib:int-sub i 1) 1)
                    (f2cl-lib:array-slice h
                                          double-float
                                          (1 1)
                                          ((1 ldh) (1 *)))
                    ldh work)))
              (if
                (<=
                  (abs
                    (f2cl-lib:fref h-%data%
                                  (k (f2cl-lib:int-sub k 1))
                                  ((1 ldh) (1 *))
                                  h-%offset%))
                    (max (* ulp tst1) smlnum))
                (go label170))))
      label70

```

```

(setf 1 k)
(cond
  (> 1 ilo)
    (setf (f2cl-lib:fref h-%data%
                        (1 (f2cl-lib:int-sub 1 1))
                        ((1 ldh) (1 *)))
          h-%offset%)
      zero)))
(if (>= 1 (f2cl-lib:int-add (f2cl-lib:int-sub i maxb) 1))
  (go label160))
(cond
  ((not wantt)
   (setf i1 1)
   (setf i2 i)))
(cond
  ((or (= its 20) (= its 30))
   (f2cl-lib:fdo (ii (f2cl-lib:int-add i (f2cl-lib:int-sub ns) 1)
                  (f2cl-lib:int-add ii 1))
                 (> ii i) nil)
   (tagbody
    (setf (f2cl-lib:fref wr-%data% (ii) ((1 *)) wr-%offset%)
          (* const
            (+
              (abs
               (f2cl-lib:fref h-%data%
                             (ii (f2cl-lib:int-sub ii 1))
                             ((1 ldh) (1 *)))
               h-%offset%))
              (abs
               (f2cl-lib:fref h-%data%
                             (ii ii)
                             ((1 ldh) (1 *)))
               h-%offset%))))))
    (setf (f2cl-lib:fref wi-%data% (ii) ((1 *)) wi-%offset%)
          zero))))
(t
 (dlacpy "Full" ns ns
  (f2cl-lib:array-slice h
                        double-float
                        ((+ i (f2cl-lib:int-sub ns) 1)
                         (f2cl-lib:int-add
                          (f2cl-lib:int-sub i ns)
                          1))
                        ((1 ldh) (1 *)))
  ldh s lds)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dlahqr nil nil ns 1 ns s lds
   (f2cl-lib:array-slice wr

```

```

double-float
((+ i (f2cl-lib:int-sub ns) 1))
((1 *)))
(f2cl-lib:array-slice wi
double-float
((+ i (f2cl-lib:int-sub ns) 1))
((1 *)))
1 ns z ldz ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
var-7 var-8 var-9 var-10 var-11 var-12))
(setf ierr var-13))
(cond
(> ierr 0)
(f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
(> ii ierr) nil)
(tagbody
(setf (f2cl-lib:fref wr-%data%
(f2cl-lib:int-add
(f2cl-lib:int-sub i ns)
ii))
((1 *))
wr-%offset%)
(f2cl-lib:fref s (ii ii) ((1 lds) (1 nsmax))))
(setf (f2cl-lib:fref wi-%data%
(f2cl-lib:int-add
(f2cl-lib:int-sub i ns)
ii))
((1 *))
wi-%offset%)
zero))))))
(setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1)))) one)
(f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
(> ii (f2cl-lib:int-add ns 1)) nil)
(tagbody
(setf (f2cl-lib:fref v (ii) ((1 (f2cl-lib:int-add nsmax 1))))
zero)))
(setf nv 1)
(f2cl-lib:fdo (j (f2cl-lib:int-add i (f2cl-lib:int-sub ns) 1)
(f2cl-lib:int-add j 1))
(> j i) nil)
(tagbody
(cond
(>= (f2cl-lib:fref wi (j) ((1 *))) zero)
(cond
(= (f2cl-lib:fref wi (j) ((1 *))) zero)
(dcopy (f2cl-lib:int-add nv 1) v 1 vv 1)
(dgemv "No transpose" (f2cl-lib:int-add nv 1) nv one
(f2cl-lib:array-slice h
double-float
(1 1)

```



```

                                ((1 ldh) (1 *)))
ldh vv 1
(- (f2cl-lib:fref wr-%data% (j) ((1 *)) wr-%offset%)) v
1)
(setf nv (f2cl-lib:int-add nv 1)))
(< (> (f2cl-lib:fref wi (j) ((1 *))) zero)
(dcopy (f2cl-lib:int-add nv 1) v 1 vv 1)
(dgemv "No transpose" (f2cl-lib:int-add nv 1) nv one
(f2cl-lib:array-slice h
                        double-float
                        (1 1)
                        ((1 ldh) (1 *)))

ldh v 1
(* (- two)
   (f2cl-lib:fref wr-%data% (j) ((1 *)) wr-%offset%))
vv 1)
(setf itemp (idamax (f2cl-lib:int-add nv 1) vv 1))
(setf temp
  (/ one
    (max
      (abs
        (f2cl-lib:fref vv
                        (itemp)
                        ((1
                          (f2cl-lib:int-add nsmax
                            1))))))
      smlnum)))
(dscal (f2cl-lib:int-add nv 1) temp vv 1)
(setf absw
  (dlapy2
    (f2cl-lib:fref wr-%data%
                    (j)
                    ((1 *))
                    wr-%offset%)
    (f2cl-lib:fref wi-%data%
                    (j)
                    ((1 *))
                    wi-%offset%)))
(setf temp (* temp absw absw))
(dgemv "No transpose" (f2cl-lib:int-add nv 2)
(f2cl-lib:int-add nv 1) one
(f2cl-lib:array-slice h
                        double-float
                        (1 1)
                        ((1 ldh) (1 *)))

ldh vv 1 temp v 1)
(setf nv (f2cl-lib:int-add nv 2)))
(setf itemp (idamax nv v 1))
(setf temp
  (abs

```

```

(f2cl-lib:fref v
  (itemp)
  ((1 (f2cl-lib:int-add nsmax 1)))))
(cond
  ((= temp zero)
    (setf (f2cl-lib:fref v
      (1)
      ((1 (f2cl-lib:int-add nsmax 1)))))
    one)
  (f2cl-lib:fdo (ii 2 (f2cl-lib:int-add ii 1))
    ((> ii nv) nil)
    (tagbody
      (setf (f2cl-lib:fref v
        (ii)
        ((1
          (f2cl-lib:int-add nsmax 1)))))
        zero))))
(t
  (setf temp (max temp smlnum))
  (dscal nv (/ one temp) v 1))))))
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  ((> k (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf nr
      (min (the fixnum (f2cl-lib:int-add ns 1))
        (the fixnum
          (f2cl-lib:int-add (f2cl-lib:int-sub i k)
            1)))))
    (if (> k 1)
      (dcopy nr
        (f2cl-lib:array-slice h
          double-float
          (k (f2cl-lib:int-sub k 1))
          ((1 ldh) (1 *)))
          1 v 1))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
        (dlarfgr nr
          (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1)))))
          (f2cl-lib:array-slice v
            double-float
            (2)
            ((1 (f2cl-lib:int-add nsmax 1)))))
          1 tau)
        (declare (ignore var-0 var-2 var-3))
        (setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1)))))
          var-1)
        (setf tau var-4))
      (cond
        ((> k 1)
          (setf (f2cl-lib:fref h-%data%

```

```

(k (f2cl-lib:int-sub k 1))
((1 ldh) (1 *))
h-%offset%)
(f2cl-lib:fref v
  (1)
  ((1 (f2cl-lib:int-add nsmax 1))))
(f2cl-lib:fdo (ii (f2cl-lib:int-add k 1)
  (f2cl-lib:int-add ii 1))
  (> ii i) nil)
(tagbody
  (setf (f2cl-lib:fref h-%data%
    (ii (f2cl-lib:int-sub k 1))
    ((1 ldh) (1 *))
    h-%offset%)
    zero))))
(setf (f2cl-lib:fref v (1) ((1 (f2cl-lib:int-add nsmax 1))))
  one)
(dlarfx "Left" nr (f2cl-lib:int-add (f2cl-lib:int-sub i2 k) 1)
  v tau
  (f2cl-lib:array-slice h double-float (k k) ((1 ldh) (1 *)))
  ldh work)
(dlarfx "Right"
  (f2cl-lib:int-add
  (f2cl-lib:int-sub
    (min (the fixnum (f2cl-lib:int-add k nr))
    (the fixnum i))
    i1)
  1)
  nr v tau
  (f2cl-lib:array-slice h double-float (i1 k) ((1 ldh) (1 *)))
  ldh work)
(cond
  (wantz
    (dlarfx "Right" nh nr v tau
      (f2cl-lib:array-slice z
        double-float
        (ilo k)
        ((1 ldz) (1 *)))
        ldz work))))))
(setf info i)
(go end_label)
label160
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
  var-10 var-11 var-12 var-13)
  (dlahqr wantt wantz n l i h ldh wr wi ilo ihi z ldz info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10 var-11 var-12))
  (setf info var-13))
(if (> info 0) (go end_label))

```

```

      (setf itn (f2cl-lib:int-sub itn its))
      (setf i (f2cl-lib:int-sub 1 1))
      (go label150)
label170
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
            (coerce
              (the fixnum
                (max (the fixnum 1)
                     (the fixnum n)))
              'double-float))
end_label
      (return
        (values nil nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

disnan LAPACK

— disnan.input —

```

)set break resume
)sys rm -f disnan.output
)spool disnan.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— disnan.help —

```

=====
dhseqr examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

LOGICAL FUNCTION DISNAN(DIN)

.. Scalar Arguments ..

DOUBLE PRECISION DIN

..

Purpose:

=====

DISNAN returns .TRUE. if its argument is NaN, and .FALSE. otherwise. To be replaced by the Fortran 2003 intrinsic in the future.

Arguments:

=====

[in] DIN

DIN is DOUBLE PRECISION

Input to test for NaN.

Authors:

=====

Univ. of Tennessee

Univ. of California Berkeley

Univ. of Colorado Denver

NAG Ltd.

November 2011

— disnan.f —

```
* =====
* LOGICAL FUNCTION DISNAN( DIN )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
* November 2011
*
* .. Scalar Arguments ..
```

```

      DOUBLE PRECISION  DIN
*      ..
*
* =====
*
* .. External Functions ..
*      LOGICAL DLAINAN
*      EXTERNAL DLAINAN
* ..
* .. Executable Statements ..
*      DISNAN = DLAINAN(DIN,DIN)
*      RETURN
*      END

```

— LAPACK disnan —

dlabad LAPACK

— dlabad.input —

```

)set break resume
)sys rm -f dlabad.output
)spool dlabad.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlabad.help —

```

=====
dlabad examples
=====

```

```
=====
Man Page Details
=====
```

NAME

DLABAD - a input the values computed by DLAMCH for underflow and overflow, and returns the square root of each of these values if the log of LARGE is sufficiently large

SYNOPSIS

```
SUBROUTINE DLABAD( SMALL, LARGE )
```

```
      DOUBLE          PRECISION LARGE, SMALL
```

PURPOSE

DLABAD takes as input the values computed by DLAMCH for underflow and overflow, and returns the square root of each of these values if the log of LARGE is sufficiently large. This subroutine is intended to identify machines with a large exponent range, such as the Crays, and redefine the underflow and overflow limits to be the square roots of the values computed by DLAMCH. This subroutine is needed because DLAMCH does not compensate for poor arithmetic in the upper half of the exponent range, as is found on a Cray.

ARGUMENTS

SMALL (input/output) DOUBLE PRECISION
On entry, the underflow threshold as computed by DLAMCH. On exit, if LOG10(LARGE) is sufficiently large, the square root of SMALL, otherwise unchanged.

LARGE (input/output) DOUBLE PRECISION
On entry, the overflow threshold as computed by DLAMCH. On exit, if LOG10(LARGE) is sufficiently large, the square root of LARGE, otherwise unchanged.

```
-----
      — dlabad.f —
```

```
SUBROUTINE DLABAD( SMALL, LARGE )
```

```
*
* -- LAPACK auxiliary routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University
* October 31, 1992
*
* .. Scalar Arguments ..
```

```

      DOUBLE PRECISION  LARGE, SMALL
*      ..
*
* =====
*
*      .. Intrinsic Functions ..
      INTRINSIC          LOG10, SQRT
*      ..
*      .. Executable Statements ..
*
*      If it looks like we're on a Cray, take the square root of
*      SMALL and LARGE to avoid overflow and underflow problems.
*
      IF( LOG10( LARGE ).GT.2000.DO ) THEN
        SMALL = SQRT( SMALL )
        LARGE = SQRT( LARGE )
      END IF
*
      RETURN
*
*      End of DLABAD
*
      END

```

— LAPACK dlabad —

```

(defun dlabad (small large)
  (declare (type (double-float) large small))
  (prog ()
    (declare)
    (cond
      ((> (f2cl-lib:log10 large) 2000.0)
       (setf small (f2cl-lib:fsqrt small))
       (setf large (f2cl-lib:fsqrt large))))
    (return (values small large))))

```

dlabrd LAPACK

— dlabrd.input —


```

)set break resume
)sys rm -f dlabrd.output
)spool dlabrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlabrd.help —

```

=====
dlabrd examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLABRD - the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation $Q' * A * P$, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A

SYNOPSIS

```
SUBROUTINE DLABRD( M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y, LDY )
```

```
      INTEGER          LDA, LDX, LDY, M, N, NB
```

```
      DOUBLE          PRECISION  A( LDA, * ), D( * ), E( * ), TAUP( * ),
      TAUQ( * ), X( LDX, * ), Y( LDY, * )
```

PURPOSE

DLABRD reduces the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation $Q' * A * P$, and returns the matrices X and Y which are needed to apply the transformation to the unreduced part of A.

If $m \geq n$, A is reduced to upper bidiagonal form; if $m < n$, to lower bidiagonal form.

This is an auxiliary routine called by DGEBRD

ARGUMENTS

- M (input) INTEGER
The number of rows in the matrix A.
- N (input) INTEGER
The number of columns in the matrix A.
- NB (input) INTEGER
The number of leading rows and columns of A to be reduced.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the m by n general matrix to be reduced. On exit, the first NB rows and columns of the matrix are overwritten; the rest of the array is unchanged. If $m \geq n$, elements on and below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors; and elements above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. If $m < n$, elements below the diagonal in the first NB columns, with the array TAUQ, represent the orthogonal matrix Q as a product of elementary reflectors, and elements on and above the diagonal in the first NB rows, with the array TAUP, represent the orthogonal matrix P as a product of elementary reflectors. See Further Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1,M)$.
- D (output) DOUBLE PRECISION array, dimension (NB)
The diagonal elements of the first NB rows and columns of the reduced matrix. $D(i) = A(i,i)$.
- E (output) DOUBLE PRECISION array, dimension (NB)
The off-diagonal elements of the first NB rows and columns of the reduced matrix.
- TAUQ (output) DOUBLE PRECISION array dimension (NB)
The scalar factors of the elementary reflectors which represent the orthogonal matrix Q. See Further Details. TAUP (output) DOUBLE PRECISION array, dimension (NB) The scalar factors of the elementary reflectors which represent the orthogonal matrix P. See Further Details. X (output) DOUBLE PRECISION array, dimension (LDX,NB) The m-by-nb matrix X required to update the unreduced part of A.
- LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq M$.
- Y (output) DOUBLE PRECISION array, dimension (LDY,NB)
The n-by-nb matrix Y required to update the unreduced part of A.

LDY (input) INTEGER
 The leading dimension of the array Y. LDY \geq N.

FURTHER DETAILS

The matrices Q and P are represented as products of elementary reflectors:

$$Q = H(1) H(2) \dots H(nb) \quad \text{and} \quad P = G(1) G(2) \dots G(nb)$$

Each H(i) and G(i) has the form:

$$H(i) = I - \tau_{uq} * v * v' \quad \text{and} \quad G(i) = I - \tau_{up} * u * u'$$

where τ_{uq} and τ_{up} are real scalars, and v and u are real vectors.

If $m \geq n$, $v(1:i-1) = 0$, $v(i) = 1$, and $v(i:m)$ is stored on exit in $A(i:m,i)$; $u(1:i) = 0$, $u(i+1) = 1$, and $u(i+1:n)$ is stored on exit in $A(i,i+1:n)$; τ_{uq} is stored in $TAUQ(i)$ and τ_{up} in $TAUP(i)$.

If $m < n$, $v(1:i) = 0$, $v(i+1) = 1$, and $v(i+1:m)$ is stored on exit in $A(i+2:m,i)$; $u(1:i-1) = 0$, $u(i) = 1$, and $u(i:n)$ is stored on exit in $A(i,i+1:n)$; τ_{uq} is stored in $TAUQ(i)$ and τ_{up} in $TAUP(i)$.

The elements of the vectors v and u together form the m-by-nb matrix V and the nb-by-n matrix U' which are needed, with X and Y, to apply the transformation to the unreduced part of the matrix, using a block update of the form: $A := A - V*Y' - X*U'$.

The contents of A on exit are illustrated by the following examples with $nb = 2$:

$m = 6$ and $n = 5$ ($m > n$):

```
( 1   1   u1  u1  u1 )
( v1  1   1   u2  u2 )
( v1  v2  a   a   a )
( v1  v2  a   a   a )
( v1  v2  a   a   a )
( v1  v2  a   a   a )
```

$m = 5$ and $n = 6$ ($m < n$):

```
( 1   u1  u1  u1  u1  u1 )
( 1   1   u2  u2  u2  u2 )
( v1  1   a   a   a   a )
( v1  v2  a   a   a   a )
( v1  v2  a   a   a   a )
```

where a denotes an element of the original matrix which is unchanged, v_i denotes an element of the vector defining H(i), and u_i an element of the vector defining G(i).

— dlabrd.f —

```

      SUBROUTINE DLABRD( M, N, NB, A, LDA, D, E, TAUQ, TAUP, X, LDX, Y,
$                      LDY )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
      INTEGER          LDA, LDX, LDY, M, N, NB
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), D( * ), E( * ), TAUP( * ),
$                      TAUQ( * ), X( LDX, * ), Y( LDY, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION  ZERO, ONE
      PARAMETER         ( ZERO = 0.0D0, ONE = 1.0D0 )
*
*  ..
*
*  .. Local Scalars ..
      INTEGER           I
*
*  ..
*
*  .. External Subroutines ..
      EXTERNAL          DGEMV, DLARFG, DSCAL
*
*  ..
*
*  .. Intrinsic Functions ..
      INTRINSIC         MIN
*
*  ..
*
*  .. Executable Statements ..
*
*  Quick return if possible
*
      IF( M.LE.0 .OR. N.LE.0 )
$    RETURN
*
      IF( M.GE.N ) THEN
*
*        Reduce to upper bidiagonal form
*
*        DO 10 I = 1, NB
*
*          Update A(i:m,i)
*
          CALL DGEMV( 'No transpose', M-I+1, I-1, -ONE, A( I, 1 ),
$                LDA, Y( I, 1 ), LDY, ONE, A( I, I ), 1 )
          CALL DGEMV( 'No transpose', M-I+1, I-1, -ONE, X( I, 1 ),
$                LDX, A( 1, I ), 1, ONE, A( I, I ), 1 )

```

```

*
*      Generate reflection Q(i) to annihilate A(i+1:m,i)
*
      CALL DLARFG( M-I+1, A( I, I ), A( MIN( I+1, M ), I ), 1,
$           TAUQ( I ) )
      D( I ) = A( I, I )
      IF( I.LT.N ) THEN
          A( I, I ) = ONE
*
*      Compute Y(i+1:n,i)
*
      CALL DGEMV( 'Transpose', M-I+1, N-I, ONE, A( I, I+1 ),
$           LDA, A( I, I ), 1, ZERO, Y( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', M-I+1, I-1, ONE, A( I, 1 ), LDA,
$           A( I, I ), 1, ZERO, Y( 1, I ), 1 )
      CALL DGEMV( 'No transpose', N-I, I-1, -ONE, Y( I+1, 1 ),
$           LDY, Y( 1, I ), 1, ONE, Y( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', M-I+1, I-1, ONE, X( I, 1 ), LDX,
$           A( I, I ), 1, ZERO, Y( 1, I ), 1 )
      CALL DGEMV( 'Transpose', I-1, N-I, -ONE, A( 1, I+1 ),
$           LDA, Y( 1, I ), 1, ONE, Y( I+1, I ), 1 )
      CALL DSCAL( N-I, TAUQ( I ), Y( I+1, I ), 1 )
*
*      Update A(i,i+1:n)
*
      CALL DGEMV( 'No transpose', N-I, I, -ONE, Y( I+1, 1 ),
$           LDY, A( I, 1 ), LDA, ONE, A( I, I+1 ), LDA )
      CALL DGEMV( 'Transpose', I-1, N-I, -ONE, A( 1, I+1 ),
$           LDA, X( I, 1 ), LDX, ONE, A( I, I+1 ), LDA )
*
*      Generate reflection P(i) to annihilate A(i,i+2:n)
*
      CALL DLARFG( N-I, A( I, I+1 ), A( I, MIN( I+2, N ) ),
$           LDA, TAUP( I ) )
      E( I ) = A( I, I+1 )
      A( I, I+1 ) = ONE
*
*      Compute X(i+1:m,i)
*
      CALL DGEMV( 'No transpose', M-I, N-I, ONE, A( I+1, I+1 ),
$           LDA, A( I, I+1 ), LDA, ZERO, X( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', N-I, I, ONE, Y( I+1, 1 ), LDY,
$           A( I, I+1 ), LDA, ZERO, X( 1, I ), 1 )
      CALL DGEMV( 'No transpose', M-I, I, -ONE, A( I+1, 1 ),
$           LDA, X( 1, I ), 1, ONE, X( I+1, I ), 1 )
      CALL DGEMV( 'No transpose', I-1, N-I, ONE, A( 1, I+1 ),
$           LDA, A( I, I+1 ), LDA, ZERO, X( 1, I ), 1 )
      CALL DGEMV( 'No transpose', M-I, I-1, -ONE, X( I+1, 1 ),
$           LDX, X( 1, I ), 1, ONE, X( I+1, I ), 1 )
      CALL DSCAL( M-I, TAUP( I ), X( I+1, I ), 1 )

```

```

      END IF
10    CONTINUE
      ELSE
*
*      Reduce to lower bidiagonal form
*
      DO 20 I = 1, NB
*
*      Update A(i,i:n)
*
      CALL DGEMV( 'No transpose', N-I+1, I-1, -ONE, Y( I, 1 ),
$          LDY, A( I, 1 ), LDA, ONE, A( I, I ), LDA )
      CALL DGEMV( 'Transpose', I-1, N-I+1, -ONE, A( 1, I ), LDA,
$          X( I, 1 ), LDX, ONE, A( I, I ), LDA )
*
*      Generate reflection P(i) to annihilate A(i,i+1:n)
*
      CALL DLARFG( N-I+1, A( I, I ), A( I, MIN( I+1, N ) ), LDA,
$          TAUP( I ) )
      D( I ) = A( I, I )
      IF( I.LT.M ) THEN
          A( I, I ) = ONE
*
*      Compute X(i+1:m,i)
*
      CALL DGEMV( 'No transpose', M-I, N-I+1, ONE, A( I+1, I ),
$          LDA, A( I, I ), LDA, ZERO, X( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', N-I+1, I-1, ONE, Y( I, 1 ), LDY,
$          A( I, I ), LDA, ZERO, X( 1, I ), 1 )
      CALL DGEMV( 'No transpose', M-I, I-1, -ONE, A( I+1, 1 ),
$          LDA, X( 1, I ), 1, ONE, X( I+1, I ), 1 )
      CALL DGEMV( 'No transpose', I-1, N-I+1, ONE, A( 1, I ),
$          LDA, A( I, I ), LDA, ZERO, X( 1, I ), 1 )
      CALL DGEMV( 'No transpose', M-I, I-1, -ONE, X( I+1, 1 ),
$          LDX, X( 1, I ), 1, ONE, X( I+1, I ), 1 )
      CALL DSCAL( M-I, TAUP( I ), X( I+1, I ), 1 )
*
*      Update A(i+1:m,i)
*
      CALL DGEMV( 'No transpose', M-I, I-1, -ONE, A( I+1, 1 ),
$          LDA, Y( I, 1 ), LDY, ONE, A( I+1, I ), 1 )
      CALL DGEMV( 'No transpose', M-I, I, -ONE, X( I+1, 1 ),
$          LDX, A( 1, I ), 1, ONE, A( I+1, I ), 1 )
*
*      Generate reflection Q(i) to annihilate A(i+2:m,i)
*
      CALL DLARFG( M-I, A( I+1, I ), A( MIN( I+2, M ), I ), 1,
$          TAUQ( I ) )
      E( I ) = A( I+1, I )
      A( I+1, I ) = ONE

```

```

*
*           Compute Y(i+1:n,i)
*
      CALL DGEMV( 'Transpose', M-I, N-I, ONE, A( I+1, I+1 ),
$           LDA, A( I+1, I ), 1, ZERO, Y( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', M-I, I-1, ONE, A( I+1, 1 ), LDA,
$           A( I+1, I ), 1, ZERO, Y( 1, I ), 1 )
      CALL DGEMV( 'No transpose', N-I, I-1, -ONE, Y( I+1, 1 ),
$           LDY, Y( 1, I ), 1, ONE, Y( I+1, I ), 1 )
      CALL DGEMV( 'Transpose', M-I, I, ONE, X( I+1, 1 ), LDX,
$           A( I+1, I ), 1, ZERO, Y( 1, I ), 1 )
      CALL DGEMV( 'Transpose', I, N-I, -ONE, A( 1, I+1 ), LDA,
$           Y( 1, I ), 1, ONE, Y( I+1, I ), 1 )
      CALL DSCAL( N-I, TAUQ( I ), Y( I+1, I ), 1 )
    END IF
20    CONTINUE
    END IF
    RETURN
*
*    End of DLABRD
*
    END

```

— LAPACK dlabrd —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlabrd (m n nb a lda d e tauq taup x ldx y ldy)
    (declare (type (simple-array double-float (*)) y x taup tauq e d a)
              (type fixnum ldy ldx lda nb n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (tauq double-float tauq-%data% tauq-%offset%)
       (taup double-float taup-%data% taup-%offset%)
       (x double-float x-%data% x-%offset%)
       (y double-float y-%data% y-%offset%))
      (prog ((i 0))
        (declare (type fixnum i))
        (if (or (<= m 0) (<= n 0)) (go end_label))
        (cond
          ((>= m n)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
             (> i nb) nil)

```

```

(tagbody
  (dgemv "No transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-sub i 1) (- one)
    (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *))) lda
    (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *))) ldy
    one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    1)
  (dgemv "No transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-sub i 1) (- one)
    (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *))) ldx
    (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1
    one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
    1)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
      (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
      (f2cl-lib:array-slice a
        double-float
        ((min (f2cl-lib:int-add i 1) m) i)
        ((1 lda) (1 *)))
      1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
    (declare (ignore var-0 var-2 var-3))
    (setf (f2cl-lib:fref a-%data%
      (i i)
      ((1 lda) (1 *))
      a-%offset%)
      var-1)
    (setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
      var-4))
  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
    (f2cl-lib:fref a-%data%
      (i i)
      ((1 lda) (1 *))
      a-%offset%))
  (cond
    ((< i n)
      (setf (f2cl-lib:fref a-%data%
        (i i)
        ((1 lda) (1 *))
        a-%offset%)
        one)
      (dgemv "Transpose"
        (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
        (f2cl-lib:int-sub n i) one
        (f2cl-lib:array-slice a
          double-float
          (i (f2cl-lib:int-add i 1))
          ((1 lda) (1 *)))

```



```

lda
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) i)
                        ((1 ldy) (1 *)))

1)
(dgemv "Transpose"
(f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
(f2cl-lib:int-sub i 1) one
(f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub n i)
(f2cl-lib:int-sub i 1) (- one)
(f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) 1)
                        ((1 ldy) (1 *)))

ldy
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1 one
(f2cl-lib:array-slice y
                        double-float
                        ((+ i 1) i)
                        ((1 ldy) (1 *)))

1)
(dgemv "Transpose"
(f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
(f2cl-lib:int-sub i 1) one
(f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *)))
ldx
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1)
(dgemv "Transpose" (f2cl-lib:int-sub i 1)
(f2cl-lib:int-sub n i) (- one)
(f2cl-lib:array-slice a
                        double-float
                        (1 (f2cl-lib:int-add i 1))
                        ((1 lda) (1 *)))

lda
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1 one
(f2cl-lib:array-slice y

```

```

double-float
((+ i 1) i)
((1 ldy) (1 *)))

1)
(dscal (f2cl-lib:int-sub n i)
(f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
(f2cl-lib:array-slice y
double-float
((+ i 1) i)
((1 ldy) (1 *)))

1)
(dgemv "No transpose" (f2cl-lib:int-sub n i) i (- one)
(f2cl-lib:array-slice y
double-float
((+ i 1) 1)
((1 ldy) (1 *)))

ldy
(f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *)))
lda one
(f2cl-lib:array-slice a
double-float
(i (f2cl-lib:int-add i 1))
((1 lda) (1 *)))

lda)
(dgemv "Transpose" (f2cl-lib:int-sub i 1)
(f2cl-lib:int-sub n i) (- one)
(f2cl-lib:array-slice a
double-float
(1 (f2cl-lib:int-add i 1))
((1 lda) (1 *)))

lda
(f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *)))
ldx one
(f2cl-lib:array-slice a
double-float
(i (f2cl-lib:int-add i 1))
((1 lda) (1 *)))

lda)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
(dlarfg (f2cl-lib:int-sub n i)
(f2cl-lib:fref a-%data%
(i (f2cl-lib:int-add i 1))
((1 lda) (1 *))
a-%offset%)
(f2cl-lib:array-slice a
double-float
(i
(min
(the fixnum
(f2cl-lib:int-add i 2))

```

```

                                (the fixnum n)))
                                ((1 lda) (1 *)))

lda
  (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *))
                    a-%offset%))

var-1)
(setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
var-4))
(setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))
(f2cl-lib:fref a-%data%
              (i (f2cl-lib:int-add i 1))
              ((1 lda) (1 *))
              a-%offset%))
(setf (f2cl-lib:fref a-%data%
                    (i (f2cl-lib:int-add i 1))
                    ((1 lda) (1 *))
                    a-%offset%))

one)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
(f2cl-lib:int-sub n i) one
(f2cl-lib:array-slice a
                      double-float
                      ((+ i 1) (f2cl-lib:int-add i 1))
                      ((1 lda) (1 *)))

lda
(f2cl-lib:array-slice a
                      double-float
                      (i (f2cl-lib:int-add i 1))
                      ((1 lda) (1 *)))

lda zero
(f2cl-lib:array-slice x
                      double-float
                      ((+ i 1) i)
                      ((1 ldx) (1 *)))

1)
(dgemv "Transpose" (f2cl-lib:int-sub n i) i one
(f2cl-lib:array-slice y
                      double-float
                      ((+ i 1) 1)
                      ((1 ldy) (1 *)))

ldy
(f2cl-lib:array-slice a
                      double-float
                      (i (f2cl-lib:int-add i 1))
                      ((1 lda) (1 *)))

lda zero

```

```

(f2cl-lib:array-slice x double-float (1 i) ((1 lda) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub m i) i (- one)
(f2cl-lib:array-slice a
double-float
((+ i 1) 1)
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice x double-float (1 i) ((1 lda) (1 *)))
1 one
(f2cl-lib:array-slice x
double-float
((+ i 1) i)
((1 lda) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub i 1)
(f2cl-lib:int-sub n i) one
(f2cl-lib:array-slice a
double-float
(1 (f2cl-lib:int-add i 1))
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice a
double-float
(i (f2cl-lib:int-add i 1))
((1 lda) (1 *)))
lda zero
(f2cl-lib:array-slice x double-float (1 i) ((1 lda) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
(f2cl-lib:int-sub i 1) (- one)
(f2cl-lib:array-slice x
double-float
((+ i 1) 1)
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice x double-float (1 i) ((1 lda) (1 *)))
1 one
(f2cl-lib:array-slice x
double-float
((+ i 1) i)
((1 lda) (1 *)))
1)
(dscal (f2cl-lib:int-sub m i)
(f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
(f2cl-lib:array-slice x
double-float
((+ i 1) i)
((1 lda) (1 *)))
1))))))

```

```

(t
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i nb) nil)
  (tagbody
    (dgemv "No transpose"
      (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
      (f2cl-lib:int-sub i 1) (- one)
      (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *))) ldy
      (f2cl-lib:array-slice a double-float (i 1) ((1 lda) (1 *))) lda
      one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
      lda)
    (dgemv "Transpose" (f2cl-lib:int-sub i 1)
      (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) (- one)
      (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) lda
      (f2cl-lib:array-slice x double-float (i 1) ((1 ldx) (1 *))) ldx
      one (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
      lda)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
        (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        (f2cl-lib:array-slice a
          double-float
          (i
            (min
              (the fixnum
                (f2cl-lib:int-add i 1))
              (the fixnum n)))
            ((1 lda) (1 *)))
          lda (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%))
      (declare (ignore var-0 var-2 var-3))
      (setf (f2cl-lib:fref a-%data%
        (i i)
        ((1 lda) (1 *))
        a-%offset%)
        var-1)
      (setf (f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
        var-4))
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref a-%data%
          (i i)
          ((1 lda) (1 *))
          a-%offset%))
    (cond
      ((< i m)
        (setf (f2cl-lib:fref a-%data%
          (i i)
          ((1 lda) (1 *))
          a-%offset%)
          one)
        (dgemv "No transpose" (f2cl-lib:int-sub m i)

```

```

(f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) one
(f2cl-lib:array-slice a
  double-float
  ((+ i 1) i)
  ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
lda zero
(f2cl-lib:array-slice x
  double-float
  ((+ i 1) i)
  ((1 ldx) (1 *)))
1)
(dgemv "Transpose"
  (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
  (f2cl-lib:int-sub i 1) one
  (f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *)))
  ldy
  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
  lda zero
  (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
  1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
  (f2cl-lib:int-sub i 1) (- one)
  (f2cl-lib:array-slice a
    double-float
    ((+ i 1) 1)
    ((1 lda) (1 *)))
  lda
  (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
  1 one
  (f2cl-lib:array-slice x
    double-float
    ((+ i 1) i)
    ((1 ldx) (1 *)))
  1)
(dgemv "No transpose" (f2cl-lib:int-sub i 1)
  (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) one
  (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *)))
  lda
  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
  lda zero
  (f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
  1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
  (f2cl-lib:int-sub i 1) (- one)
  (f2cl-lib:array-slice x
    double-float
    ((+ i 1) 1)
    ((1 ldx) (1 *)))

```

```

ldx
(f2cl-lib:array-slice x double-float (1 i) ((1 ldx) (1 *)))
1 one
(f2cl-lib:array-slice x
      double-float
      ((+ i 1) i)
      ((1 ldx) (1 *)))
1)
(dscal (f2cl-lib:int-sub m i)
(f2cl-lib:fref taup-%data% (i) ((1 *)) taup-%offset%)
(f2cl-lib:array-slice x
      double-float
      ((+ i 1) i)
      ((1 ldx) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub m i)
(f2cl-lib:int-sub i 1) (- one)
(f2cl-lib:array-slice a
      double-float
      ((+ i 1) 1)
      ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice y double-float (i 1) ((1 ldy) (1 *)))
ldy one
(f2cl-lib:array-slice a
      double-float
      ((+ i 1) i)
      ((1 lda) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub m i) i (- one)
(f2cl-lib:array-slice x
      double-float
      ((+ i 1) 1)
      ((1 ldx) (1 *)))
ldx
(f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *)))
1 one
(f2cl-lib:array-slice a
      double-float
      ((+ i 1) i)
      ((1 lda) (1 *)))
1)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlarfg (f2cl-lib:int-sub m i)
    (f2cl-lib:fref a-%data%
      ((f2cl-lib:int-add i 1) i)
      ((1 lda) (1 *))
      a-%offset%)
    (f2cl-lib:array-slice a
      double-float

```

```

((min (f2cl-lib:int-add i 2) m) i)
((1 lda) (1 *)))
1 (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data%
((f2cl-lib:int-add i 1) i)
((1 lda) (1 *))
a-%offset%)
var-1)
(setf (f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
var-4))
(setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add i 1) i)
((1 lda) (1 *))
a-%offset%))
(setf (f2cl-lib:fref a-%data%
((f2cl-lib:int-add i 1) i)
((1 lda) (1 *))
a-%offset%)
one)
(dgemv "Transpose" (f2cl-lib:int-sub m i)
(f2cl-lib:int-sub n i) one
(f2cl-lib:array-slice a
double-float
((+ i 1) (f2cl-lib:int-add i 1))
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice a
double-float
((+ i 1) i)
((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y
double-float
((+ i 1) i)
((1 ldy) (1 *)))
1)
(dgemv "Transpose" (f2cl-lib:int-sub m i)
(f2cl-lib:int-sub i 1) one
(f2cl-lib:array-slice a
double-float
((+ i 1) 1)
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice a
double-float
((+ i 1) i)
((1 lda) (1 *)))
1 zero

```



```

(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1)
(dgemv "No transpose" (f2cl-lib:int-sub n i)
(f2cl-lib:int-sub i 1) (- one)
(f2cl-lib:array-slice y
double-float
((+ i 1) 1)
((1 ldy) (1 *)))
ldy
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1 one
(f2cl-lib:array-slice y
double-float
((+ i 1) i)
((1 ldy) (1 *)))
1)
(dgemv "Transpose" (f2cl-lib:int-sub m i) i one
(f2cl-lib:array-slice x
double-float
((+ i 1) 1)
((1 ldx) (1 *)))
ldx
(f2cl-lib:array-slice a
double-float
((+ i 1) i)
((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1)
(dgemv "Transpose" i (f2cl-lib:int-sub n i) (- one)
(f2cl-lib:array-slice a
double-float
(1 (f2cl-lib:int-add i 1))
((1 lda) (1 *)))
lda
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 *)))
1 one
(f2cl-lib:array-slice y
double-float
((+ i 1) i)
((1 ldy) (1 *)))
1)
(dscal (f2cl-lib:int-sub n i)
(f2cl-lib:fref tauq-%data% (i) ((1 *)) tauq-%offset%)
(f2cl-lib:array-slice y
double-float
((+ i 1) i)
((1 ldy) (1 *)))
1))))))

```

end_label

```
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil nil)))))
```

dlacon LAPACK

— dlacon.input —

```
)set break resume
)sys rm -f dlacon.output
)spool dlacon.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlacon.help —

```
=====
dlacon examples
=====

=====
Man Page Details
=====
```

NAME

DLACON - the 1-norm of a square, real matrix A

SYNOPSIS

```
SUBROUTINE DLACON( N, V, X, ISGN, EST, KASE )
```

```
      INTEGER      KASE, N
```

```
      DOUBLE      PRECISION EST
```

```
      INTEGER      ISGN( * )
```

```
      DOUBLE      PRECISION V( * ), X( * )
```

PURPOSE

DLACON estimates the 1-norm of a square, real matrix A. Reverse communication is used for evaluating matrix-vector products.

ARGUMENTS

N (input) INTEGER
The order of the matrix. $N \geq 1$.

V (workspace) DOUBLE PRECISION array, dimension (N)
On the final return, $V = A*W$, where $EST = \text{norm}(V)/\text{norm}(W)$ (W is not returned).

X (input/output) DOUBLE PRECISION array, dimension (N)
On an intermediate return, X should be overwritten by $A * X$, if $KASE=1$, $A' * X$, if $KASE=2$, and DLACON must be re-called with all the other parameters unchanged.

ISGN (workspace) INTEGER array, dimension (N)

EST (input/output) DOUBLE PRECISION
On entry with $KASE = 1$ or 2 and $JUMP = 3$, EST should be unchanged from the previous call to DLACON. On exit, EST is an estimate (a lower bound) for $\text{norm}(A)$.

KASE (input/output) INTEGER
On the initial call to DLACON, KASE should be 0. On an intermediate return, KASE will be 1 or 2, indicating whether X should be overwritten by $A * X$ or $A' * X$. On the final return from DLACON, KASE will again be 0.

FURTHER DETAILS

Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation", ACM Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.

— dlacon.f —

SUBROUTINE DLACON(N, V, X, ISGN, EST, KASE)

```
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
```

```

      INTEGER          KASE, N
      DOUBLE PRECISION EST
*
*   ..
*   .. Array Arguments ..
      INTEGER          ISGN( * )
      DOUBLE PRECISION V( * ), X( * )
*
*   ..
*
*   =====
*
*   .. Parameters ..
      INTEGER          ITMAX
      PARAMETER        ( ITMAX = 5 )
      DOUBLE PRECISION ZERO, ONE, TWO
      PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0 )
*
*   ..
*   .. Local Scalars ..
      INTEGER          I, ITER, J, JLAST, JUMP
      DOUBLE PRECISION ALTSGN, ESTOLD, TEMP
*
*   ..
*   .. External Functions ..
      INTEGER          IDAMAX
      DOUBLE PRECISION DASUM
      EXTERNAL          IDAMAX, DASUM
*
*   ..
*   .. External Subroutines ..
      EXTERNAL          DCOPY
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC          ABS, DBLE, NINT, SIGN
*
*   ..
*   .. Save statement ..
      SAVE
*
*   ..
*   .. Executable Statements ..
*
      IF( KASE.EQ.0 ) THEN
        DO 10 I = 1, N
          X( I ) = ONE / DBLE( N )
10      CONTINUE
        KASE = 1
        JUMP = 1
        RETURN
      END IF
*
      GO TO ( 20, 40, 70, 110, 140 )JUMP
*
*   ..... ENTRY   (JUMP = 1)
*   FIRST ITERATION.  X HAS BEEN OVERWRITTEN BY A*X.
*

```

```

20 CONTINUE
  IF( N.EQ.1 ) THEN
    V( 1 ) = X( 1 )
    EST = ABS( V( 1 ) )
*    ... QUIT
*    GO TO 150
  END IF
  EST = DASUM( N, X, 1 )
*
  DO 30 I = 1, N
    X( I ) = SIGN( ONE, X( I ) )
    ISGN( I ) = NINT( X( I ) )
30 CONTINUE
  KASE = 2
  JUMP = 2
  RETURN
*
*    ..... ENTRY    (JUMP = 2)
*    FIRST ITERATION.  X HAS BEEN OVERWRITTEN BY TRANDPOSE(A)*X.
*
40 CONTINUE
  J = IDAMAX( N, X, 1 )
  ITER = 2
*
*    MAIN LOOP - ITERATIONS 2,3,...,ITMAX.
*
50 CONTINUE
  DO 60 I = 1, N
    X( I ) = ZERO
60 CONTINUE
  X( J ) = ONE
  KASE = 1
  JUMP = 3
  RETURN
*
*    ..... ENTRY    (JUMP = 3)
*    X HAS BEEN OVERWRITTEN BY A*X.
*
70 CONTINUE
  CALL DCOPY( N, X, 1, V, 1 )
  ESTOLD = EST
  EST = DASUM( N, V, 1 )
  DO 80 I = 1, N
    IF( NINT( SIGN( ONE, X( I ) ) ) .NE. ISGN( I ) )
$      GO TO 90
80 CONTINUE
*    REPEATED SIGN VECTOR DETECTED, HENCE ALGORITHM HAS CONVERGED.
*    GO TO 120
*
90 CONTINUE

```

```

*      TEST FOR CYCLING.
      IF( EST.LE.ESTOLD )
$      GO TO 120
*
      DO 100 I = 1, N
          X( I ) = SIGN( ONE, X( I ) )
          ISGN( I ) = NINT( X( I ) )
100 CONTINUE
      KASE = 2
      JUMP = 4
      RETURN

*
*      ..... ENTRY    (JUMP = 4)
*      X HAS BEEN OVERWRITTEN BY TRANDPOSE(A)*X.
*
110 CONTINUE
      JLAST = J
      J = IDAMAX( N, X, 1 )
      IF( ( X( JLAST ).NE.ABS( X( J ) ) ) .AND. ( ITER.LT.ITMAX ) ) THEN
          ITER = ITER + 1
          GO TO 50
      END IF

*
*      ITERATION COMPLETE.  FINAL STAGE.
*
120 CONTINUE
      ALTSGN = ONE
      DO 130 I = 1, N
          X( I ) = ALTSGN*( ONE+DBLE( I-1 ) / DBLE( N-1 ) )
          ALTSGN = -ALTSGN
130 CONTINUE
      KASE = 1
      JUMP = 5
      RETURN

*
*      ..... ENTRY    (JUMP = 5)
*      X HAS BEEN OVERWRITTEN BY A*X.
*
140 CONTINUE
      TEMP = TWO*( DASUM( N, X, 1 ) / DBLE( 3*N ) )
      IF( TEMP.GT.EST ) THEN
          CALL DCOPY( N, X, 1, V, 1 )
          EST = TEMP
      END IF

*
150 CONTINUE
      KASE = 0
      RETURN

*
*      End of DLACON

```

*

END

— LAPACK dlacon —

```
(let* ((itmax 5) (zero 0.0) (one 1.0) (two 2.0))
  (declare (type (fixnum 5 5) itmax)
            (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two))
  (let ((altsgn 0.0)
        (estold 0.0)
        (temp 0.0)
        (i 0)
        (iter 0)
        (j 0)
        (jlast 0)
        (jump 0))
    (declare (type fixnum itmax jump jlast j iter i)
              (type (double-float) two one zero temp estold altsgn))
    (defun dlacon (n v x isgn est kase)
      (declare (type (double-float) est)
                (type (simple-array fixnum (*)) isgn)
                (type (simple-array double-float (*)) x v)
                (type fixnum kase n))
      (f2cl-lib:with-multi-array-data
        ((v double-float v-%data% v-%offset%)
         (x double-float x-%data% x-%offset%)
         (isgn fixnum isgn-%data% isgn-%offset%))
        (prog ()
          (declare)
          (cond
            ((= kase 0)
             (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
                           (> i n) nil)
             (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                    (/ one (coerce (realpart n) 'double-float))))
              (setf kase 1)
              (setf jump 1)
              (go end_label)))
            (t
             (f2cl-lib:computed-goto (label20 label40 label70 label110 label140)
                                     jump)
             label20
             (cond
              ((= n 1)
```

```

      (setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
            (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%))
      (setf est (abs (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)))
      (go label150)))
(setf est (dasum n x 1))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
        (f2cl-lib:sign one
          (f2cl-lib:fref x-%data%
            (i)
            ((1 *))
            x-%offset%)))
  (setf (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%)
        (values (round
          (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%))))))
(setf kase 2)
(setf jump 2)
(go end_label)
label40
  (setf j (idamax n x 1))
  (setf iter 2)
label50
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%) zero)))
  (setf (f2cl-lib:fref x-%data% (j) ((1 *)) x-%offset%) one)
  (setf kase 1)
  (setf jump 3)
  (go end_label)
label70
  (dcopy n x 1 v 1)
  (setf estold est)
  (setf est (dasum n v 1))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (if
      (/=
        (values (round
          (f2cl-lib:sign one
            (f2cl-lib:fref x-%data%
              (i)
              ((1 *))
              x-%offset%))))
          (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%))
      (go label90))))
  (go label120)

```



```

label190
  (if (<= est estold) (go label120))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
      (f2cl-lib:sign one
        (f2cl-lib:fref x-%data%
          (i)
          ((1 *))
          x-%offset%)))
    (setf (f2cl-lib:fref isgn-%data% (i) ((1 *)) isgn-%offset%)
      (values (round
        (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%))))))
  (setf kase 2)
  (setf jump 4)
  (go end_label)
label110
  (setf jlast j)
  (setf j (idamax n x 1))
  (cond
    ((and
      (/= (f2cl-lib:fref x (jlast) ((1 *)))
        (abs (f2cl-lib:fref x (j) ((1 *))))))
      (< iter itmax))
      (setf iter (f2cl-lib:int-add iter 1))
      (go label150)))
label120
  (setf altsgn one)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
      (* altsgn
        (+ one
          (/
            (coerce (realpart (f2cl-lib:int-sub i 1)) 'double-float)
            (coerce (realpart (f2cl-lib:int-sub n 1)) 'double-float))))))
    (setf altsgn (- altsgn)))
  (setf kase 1)
  (setf jump 5)
  (go end_label)
label140
  (setf temp
    (* two
      (/ (dasum n x 1)
        (coerce (realpart (f2cl-lib:int-mul 3 n)) 'double-float))))
  (cond
    (> temp est)
    (dcopy n x 1 v 1)

```

```

                (setf est temp)))
label150
    (setf kase 0)
end_label
    (return (values nil nil nil nil est kase))))))

```

dlacpy LAPACK

— dlacpy.input —

```

)set break resume
)sys rm -f dlacpy.output
)spool dlacpy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlacpy.help —

```

=====
dlacpy examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLACPY - all or part of a two-dimensional matrix A to another matrix B

SYNOPSIS

```
SUBROUTINE DLACPY( UPLO, M, N, A, LDA, B, LDB )
```

CHARACTER UPLO

INTEGER LDA, LDB, M, N

DOUBLE PRECISION A(LDA, *), B(LDB, *)

PURPOSE

DLACPY copies all or part of a two-dimensional matrix A to another matrix B.

ARGUMENTS

UPLO (input) CHARACTER*1
 Specifies the part of the matrix A to be copied to B. = 'U':
 Upper triangular part
 = 'L': Lower triangular part
 Otherwise: All of the matrix A

M (input) INTEGER
 The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix A. $N \geq 0$.

A (input) DOUBLE PRECISION array, dimension (LDA,N)
 The m by n matrix A. If UPLO = 'U', only the upper triangle or trapezoid is accessed; if UPLO = 'L', only the lower triangle or trapezoid is accessed.

LDA (input) INTEGER
 The leading dimension of the array A. $LDA \geq \max(1,M)$.

B (output) DOUBLE PRECISION array, dimension (LDB,N)
 On exit, $B = A$ in the locations specified by UPLO.

LDB (input) INTEGER
 The leading dimension of the array B. $LDB \geq \max(1,M)$.

— dlacpy.f —

SUBROUTINE DLACPY(UPLO, M, N, A, LDA, B, LDB)

```
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
*  CHARACTER          UPLO
*  INTEGER            LDA, LDB, M, N
*  ..
```

```

*      .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), B( LDB, * )
*      ..
*
*      =====
*
*      .. Local Scalars ..
      INTEGER           I, J
*      ..
*      .. External Functions ..
      LOGICAL           LSAME
      EXTERNAL          LSAME
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC         MIN
*      ..
*      .. Executable Statements ..
*
      IF( LSAME( UPLO, 'U' ) ) THEN
        DO 20 J = 1, N
          DO 10 I = 1, MIN( J, M )
            B( I, J ) = A( I, J )
10          CONTINUE
20        CONTINUE
      ELSE IF( LSAME( UPLO, 'L' ) ) THEN
        DO 40 J = 1, N
          DO 30 I = J, M
            B( I, J ) = A( I, J )
30          CONTINUE
40        CONTINUE
      ELSE
        DO 60 J = 1, N
          DO 50 I = 1, M
            B( I, J ) = A( I, J )
50          CONTINUE
60        CONTINUE
      END IF
      RETURN
*
*      End of DLACPY
*
      END

```

— LAPACK dlacpy —

```
(defun dlacpy (uplo m n a lda b ldb$)
```

```

(declare (type (simple-array double-float (*)) b a)
  (type fixnum ldb$ lda n m)
  (type character uplo))
(f2cl-lib:with-multi-array-data
  ((uplo character uplo-%data% uplo-%offset%)
   (a double-float a-%data% a-%offset%)
   (b double-float b-%data% b-%offset%))
  (prog ((i 0) (j 0))
    (declare (type fixnum j i))
    (cond
      ((char-equal uplo #\U)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j n) nil)
          (tagbody
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i
                (min (the fixnum j)
                     (the fixnum m)))
                nil)
              (tagbody
                (setf (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%)
                      (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)))))))
        ((char-equal uplo #\L)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            ((> j n) nil)
            (tagbody
              (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
                ((> i m) nil)
                (tagbody
                  (setf (f2cl-lib:fref b-%data%
                                       (i j)
                                       ((1 ldb$) (1 *))
                                       b-%offset%)
                        (f2cl-lib:fref a-%data%
                                       (i j)
                                       ((1 lda) (1 *))
                                       a-%offset%)))))))
          (t
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j n) nil)
              (tagbody
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i m) nil)
                  (tagbody

```

```

                (setf (f2cl-lib:fref b-%data%
                                   (i j)
                                   ((1 ldb$) (1 *))
                                   b-%offset%)
                      (f2cl-lib:fref a-%data%
                                   (i j)
                                   ((1 lda) (1 *))
                                   a-%offset%)))))))))
end_label
(return (values nil nil nil nil nil nil nil))))

```

dladiv LAPACK

— dladiv.input —

```

)set break resume
)sys rm -f dladiv.output
)spool dladiv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dladiv.help —

```

=====
dladiv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLADIV performs complex division in real arithmetic

$$p + i*q = \frac{a + i*b}{c + i*d}$$

The algorithm is due to Robert L. Smith and can be found
in D. Knuth, The art of Computer Programming, Vol.2, p.195

Arguments

=====

A (input) DOUBLE PRECISION
B (input) DOUBLE PRECISION
C (input) DOUBLE PRECISION
D (input) DOUBLE PRECISION
 The scalars a, b, c, and d in the above expression.

P (output) DOUBLE PRECISION
Q (output) DOUBLE PRECISION
 The scalars p and q in the above expression.

— dladiv.f —

```

SUBROUTINE DLADIV( A, B, C, D, P, Q )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1992
*
*  .. Scalar Arguments ..
      DOUBLE PRECISION  A, B, C, D, P, Q
*  ..
*
*  =====
*
*  .. Local Scalars ..
      DOUBLE PRECISION  E, F
*  ..
*  .. Intrinsic Functions ..
      INTRINSIC          ABS
*  ..
*  .. Executable Statements ..
*
      IF( ABS( D ).LT.ABS( C ) ) THEN
         E = D / C
         F = C + D*E
         P = ( A+B*E ) / F
         Q = ( B-A*E ) / F
      ELSE

```

```

      E = C / D
      F = D + C*E
      P = ( B+A*E ) / F
      Q = ( -A+B*E ) / F
END IF
*
      RETURN
*
*      End of DLADIV
*
      END

```

— LAPACK dladiv —

```

(defun dladiv (a b c d p q)
  (declare (type (double-float) q p d c b a))
  (prog ((e 0.0) (f 0.0))
    (declare (type (double-float) f e))
    (cond
      ((< (abs d) (abs c))
        (setf e (/ d c))
        (setf f (+ c (* d e)))
        (setf p (/ (+ a (* b e)) f))
        (setf q (/ (- b (* a e)) f)))
      (t
        (setf e (/ c d))
        (setf f (+ d (* c e)))
        (setf p (/ (+ b (* a e)) f))
        (setf q (/ (- (* b e) a) f))))
    (return (values nil nil nil nil p q))))

```

dlaed6 LAPACK

— dlaed6.input —

```

)set break resume
)sys rm -f dlaed6.output
)spool dlaed6.output
)set message test on
)set message auto off

```



```
)clear all
```

```
)spool
```

```
)lisp (bye)
```

— dlaed6.help —

```
=====
dlaed6 examples
=====
```

```
=====
Man Page Details
=====
```

Purpose

```
=====
```

DLAED6 computes the positive or negative root (closest to the origin) of

$$f(x) = \text{rho} + \frac{z(1)}{d(1)-x} + \frac{z(2)}{d(2)-x} + \frac{z(3)}{d(3)-x}$$

It is assumed that

if ORGATI = .true. the root is between d(2) and d(3);
otherwise it is between d(1) and d(2)

This routine will be called by DLAED4 when necessary. In most cases, the root sought is the smallest in magnitude, though it might not be in some extremely rare situations.

Arguments

```
=====
```

KNITER (input) INTEGER
 Refer to DLAED4 for its significance.

ORGATI (input) LOGICAL
 If ORGATI is true, the needed root is between d(2) and d(3); otherwise it is between d(1) and d(2). See DLAED4 for further details.

RHO (input) DOUBLE PRECISION
 Refer to the equation f(x) above.

D (input) DOUBLE PRECISION array, dimension (3)
D satisfies $d(1) < d(2) < d(3)$.

Z (input) DOUBLE PRECISION array, dimension (3)
Each of the elements in z must be positive.

FINIT (input) DOUBLE PRECISION
The value of f at 0. It is more accurate than the one
evaluated inside this routine (if someone wants to do
so).

TAU (output) DOUBLE PRECISION
The root of the equation $f(x)$.

INFO (output) INTEGER
= 0: successful exit
> 0: if INFO = 1, failure to converge

Further Details
=====

Based on contributions by
Ren-Cang Li, Computer Science Division, University of California
at Berkeley, USA

— dlaed6.f —

```

SUBROUTINE DLAED6( KNITER, ORGATI, RHO, D, Z, FINIT, TAU, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*  Courant Institute, NAG Ltd., and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      LOGICAL      ORGATI
      INTEGER      INFO, KNITER
      DOUBLE PRECISION  FINIT, RHO, TAU
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION  D( 3 ), Z( 3 )
*
*  ..
*
*  =====
*
*  .. Parameters ..

```

```

      INTEGER          MAXIT
      PARAMETER        ( MAXIT = 20 )
      DOUBLE PRECISION ZERO, ONE, TWO, THREE, FOUR, EIGHT
      PARAMETER        ( ZERO = 0.0D0, ONE = 1.0D0, TWO = 2.0D0,
$                       THREE = 3.0D0, FOUR = 4.0D0, EIGHT = 8.0D0 )
*
*      ..
*      .. External Functions ..
      DOUBLE PRECISION DLAMCH
      EXTERNAL          DLAMCH
*
*      ..
*      .. Local Arrays ..
      DOUBLE PRECISION DSCALE( 3 ), ZSCALE( 3 )
*
*      ..
*      .. Local Scalars ..
      LOGICAL          FIRST, SCALE
      INTEGER          I, ITER, NITER
      DOUBLE PRECISION A, B, BASE, C, DDF, DF, EPS, ERRETM, ETA, F,
$                     FC, SCLFAC, SCLINV, SMALL1, SMALL2, SMINV1,
$                     SMINV2, TEMP, TEMP1, TEMP2, TEMP3, TEMP4
*
*      ..
*      .. Save statement ..
      SAVE             FIRST, SMALL1, SMINV1, SMALL2, SMINV2, EPS
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC        ABS, INT, LOG, MAX, MIN, SQRT
*
*      ..
*      .. Data statements ..
      DATA             FIRST / .TRUE. /
*
*      ..
*      .. Executable Statements ..
*
      INFO = 0
*
      NITER = 1
      TAU = ZERO
      IF( KNITER.EQ.2 ) THEN
        IF( ORGATI ) THEN
          TEMP = ( D( 3 )-D( 2 ) ) / TWO
          C = RHO + Z( 1 ) / ( ( D( 1 )-D( 2 ) )-TEMP )
          A = C*( D( 2 )+D( 3 ) ) + Z( 2 ) + Z( 3 )
          B = C*D( 2 )*D( 3 ) + Z( 2 )*D( 3 ) + Z( 3 )*D( 2 )
        ELSE
          TEMP = ( D( 1 )-D( 2 ) ) / TWO
          C = RHO + Z( 3 ) / ( ( D( 3 )-D( 2 ) )-TEMP )
          A = C*( D( 1 )+D( 2 ) ) + Z( 1 ) + Z( 2 )
          B = C*D( 1 )*D( 2 ) + Z( 1 )*D( 2 ) + Z( 2 )*D( 1 )
        END IF
      TEMP = MAX( ABS( A ), ABS( B ), ABS( C ) )
      A = A / TEMP
      B = B / TEMP

```

```

      C = C / TEMP
      IF( C.EQ.ZERO ) THEN
        TAU = B / A
      ELSE IF( A.LE.ZERO ) THEN
        TAU = ( A-SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
      ELSE
        TAU = TWO*B / ( A+SQRT( ABS( A*A-FOUR*B*C ) ) )
      END IF
      TEMP = RHO + Z( 1 ) / ( D( 1 )-TAU ) +
$      Z( 2 ) / ( D( 2 )-TAU ) + Z( 3 ) / ( D( 3 )-TAU )
      IF( ABS( FINIT ).LE.ABS( TEMP ) )
$      TAU = ZERO
      END IF
*
*   On first call to routine, get machine parameters for
*   possible scaling to avoid overflow
*
      IF( FIRST ) THEN
        EPS = DLAMCH( 'Epsilon' )
        BASE = DLAMCH( 'Base' )
        SMALL1 = BASE**( INT( LOG( DLAMCH( 'SafMin' ) ) / LOG( BASE ) ) /
$        THREE ) )
        SMINV1 = ONE / SMALL1
        SMALL2 = SMALL1*SMALL1
        SMINV2 = SMINV1*SMINV1
        FIRST = .FALSE.
      END IF
*
*   Determine if scaling of inputs necessary to avoid overflow
*   when computing 1/TEMP**3
*
      IF( ORGATI ) THEN
        TEMP = MIN( ABS( D( 2 )-TAU ), ABS( D( 3 )-TAU ) )
      ELSE
        TEMP = MIN( ABS( D( 1 )-TAU ), ABS( D( 2 )-TAU ) )
      END IF
      SCALE = .FALSE.
      IF( TEMP.LE.SMALL1 ) THEN
        SCALE = .TRUE.
        IF( TEMP.LE.SMALL2 ) THEN
*
*   Scale up by power of radix nearest 1/SAFMIN**(2/3)
*
          SCLFAC = SMINV2
          SCLINV = SMALL2
        ELSE
*
*   Scale up by power of radix nearest 1/SAFMIN**(1/3)
*
          SCLFAC = SMINV1

```

```

        SCLINV = SMALL1
    END IF
*
*   Scaling up safe because D, Z, TAU scaled elsewhere to be 0(1)
*
    DO 10 I = 1, 3
        DSCALE( I ) = D( I )*SCLFAC
        ZSCALE( I ) = Z( I )*SCLFAC
10    CONTINUE
        TAU = TAU*SCLFAC
    ELSE
*
*   Copy D and Z to DSCALE and ZSCALE
*
    DO 20 I = 1, 3
        DSCALE( I ) = D( I )
        ZSCALE( I ) = Z( I )
20    CONTINUE
    END IF
*
    FC = ZERO
    DF = ZERO
    DDF = ZERO
    DO 30 I = 1, 3
        TEMP = ONE / ( DSCALE( I )-TAU )
        TEMP1 = ZSCALE( I )*TEMP
        TEMP2 = TEMP1*TEMP
        TEMP3 = TEMP2*TEMP
        FC = FC + TEMP1 / DSCALE( I )
        DF = DF + TEMP2
        DDF = DDF + TEMP3
30    CONTINUE
        F = FINIT + TAU*FC
*
        IF( ABS( F ).LE.ZERO )
            $    GO TO 60
*
*   Iteration begins
*
*   It is not hard to see that
*
*       1) Iterations will go up monotonically
*          if FINIT < 0;
*
*       2) Iterations will go down monotonically
*          if FINIT > 0.
*
        ITER = NITER + 1
*
    DO 50 NITER = ITER, MAXIT

```

```

*
      IF( ORGATI ) THEN
         TEMP1 = DSCALE( 2 ) - TAU
         TEMP2 = DSCALE( 3 ) - TAU
      ELSE
         TEMP1 = DSCALE( 1 ) - TAU
         TEMP2 = DSCALE( 2 ) - TAU
      END IF
      A = ( TEMP1+TEMP2 )*F - TEMP1*TEMP2*DF
      B = TEMP1*TEMP2*F
      C = F - ( TEMP1+TEMP2 )*DF + TEMP1*TEMP2*DDF
      TEMP = MAX( ABS( A ), ABS( B ), ABS( C ) )
      A = A / TEMP
      B = B / TEMP
      C = C / TEMP
      IF( C.EQ.ZERO ) THEN
         ETA = B / A
      ELSE IF( A.LE.ZERO ) THEN
         ETA = ( A-SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
      ELSE
         ETA = TWO*B / ( A+SQRT( ABS( A*A-FOUR*B*C ) ) )
      END IF
      IF( F*ETA.GE.ZERO ) THEN
         ETA = -F / DF
      END IF

*
      TEMP = ETA + TAU
      IF( ORGATI ) THEN
         IF( ETA.GT.ZERO .AND. TEMP.GE.DSCALE( 3 ) )
$           ETA = ( DSCALE( 3 )-TAU ) / TWO
         IF( ETA.LT.ZERO .AND. TEMP.LE.DSCALE( 2 ) )
$           ETA = ( DSCALE( 2 )-TAU ) / TWO
         ELSE
            IF( ETA.GT.ZERO .AND. TEMP.GE.DSCALE( 2 ) )
$              ETA = ( DSCALE( 2 )-TAU ) / TWO
            IF( ETA.LT.ZERO .AND. TEMP.LE.DSCALE( 1 ) )
$              ETA = ( DSCALE( 1 )-TAU ) / TWO
            END IF
            TAU = TAU + ETA
         END IF

*
      FC = ZERO
      ERRETM = ZERO
      DF = ZERO
      DDF = ZERO
      DO 40 I = 1, 3
         TEMP = ONE / ( DSCALE( I )-TAU )
         TEMP1 = ZSCALE( I )*TEMP
         TEMP2 = TEMP1*TEMP
         TEMP3 = TEMP2*TEMP
         TEMP4 = TEMP1 / DSCALE( I )

```

```

        FC = FC + TEMP4
        ERRETM = ERRETM + ABS( TEMP4 )
        DF = DF + TEMP2
        DDF = DDF + TEMP3
40     CONTINUE
        F = FINIT + TAU*FC
        ERRETM = EIGHT*( ABS( FINIT )+ABS( TAU )*ERRETM ) +
$       ABS( TAU )*DF
        IF( ABS( F ).LE.EPS*ERRETM )
$       GO TO 60
50     CONTINUE
        INFO = 1
60     CONTINUE
*
*     Undo scaling
*
        IF( SCALE )
$     TAU = TAU*SCLINV
        RETURN
*
*     End of DLAED6
*
        END

```

— LAPACK dlaed6 —

```

(let* ((maxit 20)
      (zero 0.0)
      (one 1.0)
      (two 2.0)
      (three 3.0)
      (four 4.0)
      (eight 8.0))
(declare (type (fixnum 20 20) maxit)
         (type (double-float 0.0 0.0) zero)
         (type (double-float 1.0 1.0) one)
         (type (double-float 2.0 2.0) two)
         (type (double-float 3.0 3.0) three)
         (type (double-float 4.0 4.0) four)
         (type (double-float 8.0 8.0) eight))
(let ((small1 0.0)
      (sminv1 0.0)
      (small12 0.0)
      (sminv2 0.0)
      (eps 0.0)
      (first$ nil))

```

```

(declare (type (member t nil) first$)
  (type (double-float) eps sminv2 small2 sminv1 small1))
(setq first$ t)
(defun dlaed6 (kniter orgati rho d z finit tau info)
  (declare (type (simple-array double-float (*)) z d)
    (type (double-float) tau finit rho)
    (type (member t nil) orgati)
    (type fixnum info kniter))
  (f2cl-lib:with-multi-array-data
    ((d double-float d-%data% d-%offset%)
     (z double-float z-%data% z-%offset%))
    (prog ((a 0.0) (b 0.0) (base 0.0) (c 0.0) (ddf 0.0) (df 0.0)
      (erretm 0.0) (eta 0.0) (f 0.0) (fc 0.0) (sclfac 0.0)
      (sclinv 0.0) (temp 0.0) (temp1 0.0) (temp2 0.0) (temp3 0.0)
      (temp4 0.0) (i 0) (iter 0) (niter 0) (scale nil)
      (dscale (make-array 3 :element-type 'double-float))
      (zscale (make-array 3 :element-type 'double-float)))
      (declare (type (double-float) a b base c ddf df erretm eta f fc
        sclfac sclinv temp temp1 temp2 temp3
        temp4)
        (type fixnum i iter niter)
        (type (member t nil) scale)
        (type (simple-array double-float (3)) dscale zscale))
      (setf info 0)
      (setf niter 1)
      (setf tau zero)
      (cond
        ((= kniter 2)
          (cond
            (orgati
              (setf temp
                (/
                  (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
                     (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
                  two))
              (setf c
                (+ rho
                  (/ (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
                     (-
                      (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
                      (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                      temp))))))
            (setf a
              (+
                (* c
                  (+ (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
                     (f2cl-lib:fref d-%data%
                      (3)
                      ((1 3))
                      d-%offset%))))))

```



```

(f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
(f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%))
(setf b
  (+
    (* c
      (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
      (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%))
    (* (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
      (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%))
    (* (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)
      (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))))))
(t
  (setf temp
    (/
      (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
        (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
      two))
  (setf c
    (+ rho
      (/ (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%)
        (-
          (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%)
          (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
          temp))))))
  (setf a
    (+
      (* c
        (+ (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
          (f2cl-lib:fref d-%data%
            (2)
            ((1 3))
            d-%offset%)))
        (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
        (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)))
    (setf b
      (+
        (* c
          (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%)
          (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
          (* (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%)
            (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%))
          (* (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%)
            (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%))))))
      (setf temp (max (abs a) (abs b) (abs c)))
      (setf a (/ a temp))
      (setf b (/ b temp))
      (setf c (/ c temp))
      (cond
        ((= c zero)
          (setf tau (/ b a)))

```

```

((<= a zero)
  (setf tau
    (/
      (- a
        (f2cl-lib:fsqrt
          (abs (+ (* a a) (* (- four) b c))))))
      (* two c))))
(t
  (setf tau
    (/ (* two b)
      (+ a
        (f2cl-lib:fsqrt
          (abs (+ (* a a) (* (- four) b c))))))))
(setf temp
  (+ rho
    (/ (f2cl-lib:fref z-%data% (1) ((1 3)) z-%offset%
      (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%
        tau))
      (/ (f2cl-lib:fref z-%data% (2) ((1 3)) z-%offset%
        (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%
          tau))
      (/ (f2cl-lib:fref z-%data% (3) ((1 3)) z-%offset%
        (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%
          tau))))
    (if (<= (abs finit) (abs temp)) (setf tau zero))))
(cond
  (first$
    (setf eps (dlamch "Epsilon"))
    (setf base (dlamch "Base"))
    (setf small1
      (expt base
        (f2cl-lib:int
          (/
            (/ (f2cl-lib:flog (dlamch "SafMin"))
              (f2cl-lib:flog base))
            three))))
    (setf sminv1 (/ one small1))
    (setf small2 (* small1 small1))
    (setf sminv2 (* sminv1 sminv1))
    (setf first$ nil)))
(cond
  (orgati
    (setf temp
      (min
        (abs
          (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%
            tau))
        (abs
          (- (f2cl-lib:fref d-%data% (3) ((1 3)) d-%offset%
            tau))))))
  (t

```

```

      (setf temp
        (min
          (abs
            (- (f2cl-lib:fref d-%data% (1) ((1 3)) d-%offset%) tau))
          (abs
            (- (f2cl-lib:fref d-%data% (2) ((1 3)) d-%offset%)
              tau))))))
    (setf scale nil)
    (cond
      ((<= temp small1)
        (setf scale t)
        (cond
          ((<= temp small2)
            (setf sclfac sminv2)
            (setf sclinv small2))
          (t
            (setf sclfac sminv1)
            (setf sclinv small1)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i 3) nil)
        (tagbody
          (setf (f2cl-lib:fref dscale (i) ((1 3)))
            (* (f2cl-lib:fref d-%data% (i) ((1 3)) d-%offset%)
              sclfac))
          (setf (f2cl-lib:fref zscale (i) ((1 3)))
            (* (f2cl-lib:fref z-%data% (i) ((1 3)) z-%offset%)
              sclfac))))
        (setf tau (* tau sclfac)))
      (t
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i 3) nil)
        (tagbody
          (setf (f2cl-lib:fref dscale (i) ((1 3)))
            (f2cl-lib:fref d-%data% (i) ((1 3)) d-%offset%))
          (setf (f2cl-lib:fref zscale (i) ((1 3)))
            (f2cl-lib:fref z-%data% (i) ((1 3)) z-%offset%))))))
    (setf fc zero)
    (setf df zero)
    (setf ddf zero)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i 3) nil)
    (tagbody
      (setf temp (/ one (- (f2cl-lib:fref dscale (i) ((1 3))) tau)))
      (setf temp1 (* (f2cl-lib:fref zscale (i) ((1 3))) temp))
      (setf temp2 (* temp1 temp))
      (setf temp3 (* temp2 temp))
      (setf fc (+ fc (/ temp1 (f2cl-lib:fref dscale (i) ((1 3)))))
      (setf df (+ df temp2))
      (setf ddf (+ ddf temp3)))
    (setf f (+ finit (* tau fc)))

```

```

(if (<= (abs f) zero) (go label60))
(setf iter (f2cl-lib:int-add niter 1))
(f2cl-lib:fdo (niter iter (f2cl-lib:int-add niter 1))
  (> niter maxit) nil)
(tagbody
  (cond
    (orgati
      (setf temp1 (- (f2cl-lib:fref dscale (2) ((1 3))) tau))
      (setf temp2 (- (f2cl-lib:fref dscale (3) ((1 3))) tau)))
    (t
      (setf temp1 (- (f2cl-lib:fref dscale (1) ((1 3))) tau))
      (setf temp2 (- (f2cl-lib:fref dscale (2) ((1 3))) tau)))
    (setf a (+ (* (+ temp1 temp2) f) (* (- temp1) temp2 df)))
    (setf b (* temp1 temp2 f))
    (setf c (+ (- f (* (+ temp1 temp2) df)) (* temp1 temp2 ddf)))
    (setf temp (max (abs a) (abs b) (abs c)))
    (setf a (/ a temp))
    (setf b (/ b temp))
    (setf c (/ c temp))
    (cond
      ((= c zero)
        (setf eta (/ b a)))
      ((<= a zero)
        (setf eta
          (/
            (- a
              (f2cl-lib:fsqrt
                (abs (+ (* a a) (* (- four) b c))))
            (* two c))))
      (t
        (setf eta
          (/ (* two b)
            (+ a
              (f2cl-lib:fsqrt
                (abs (+ (* a a) (* (- four) b c))))))))
    (cond
      ((>= (* f eta) zero)
        (setf eta (/ (- f) df)))
      (setf temp (+ eta tau))
      (cond
        (orgati
          (if
            (and (> eta zero)
              (>= temp (f2cl-lib:fref dscale (3) ((1 3)))))
            (setf eta (/ (- (f2cl-lib:fref dscale (3) ((1 3))) tau) two)))
          (if
            (and (< eta zero)
              (<= temp (f2cl-lib:fref dscale (2) ((1 3)))))
            (setf eta
              (/ (- (f2cl-lib:fref dscale (2) ((1 3))) tau) two))))

```

```

(t
  (if
    (and (> eta zero)
      (>= temp (f2cl-lib:fref dscale (2) ((1 3)))))
    (setf eta (/ (- (f2cl-lib:fref dscale (2) ((1 3)) tau) two)))
    (if
      (and (< eta zero)
        (<= temp (f2cl-lib:fref dscale (1) ((1 3)))))
      (setf eta
        (/ (- (f2cl-lib:fref dscale (1) ((1 3)) tau)
            two))))
    (setf tau (+ tau eta))
    (setf fc zero)
    (setf erretm zero)
    (setf df zero)
    (setf ddf zero)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i 3) nil)
      (tagbody
        (setf temp
          (/ one (- (f2cl-lib:fref dscale (i) ((1 3)) tau)))
        (setf temp1 (* (f2cl-lib:fref zscale (i) ((1 3)) temp))
        (setf temp2 (* temp1 temp))
        (setf temp3 (* temp2 temp))
        (setf temp4 (/ temp1 (f2cl-lib:fref dscale (i) ((1 3)))))
        (setf fc (+ fc temp4))
        (setf erretm (+ erretm (abs temp4)))
        (setf df (+ df temp2))
        (setf ddf (+ ddf temp3)))
        (setf f (+ finit (* tau fc)))
        (setf erretm
          (+ (* eight (+ (abs finit) (* (abs tau) erretm))
              (* (abs tau) df)))
          (if (<= (abs f) (* eps erretm)) (go label60))))
      (setf info 1)
    label60
      (if scale (setf tau (* tau sclinv)))
    end_label
      (return (values nil nil nil nil nil nil tau info))))))

```

dlaexc LAPACK

— dlaexc.input —

```

)set break resume
)sys rm -f dlaexc.output
)spool dlaexc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlaexc.help —

```

=====
dlaexc examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLAEXC - adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation

SYNOPSIS

```
SUBROUTINE DLAEXC( WANTQ, N, T, LDT, Q, LDQ, J1, N1, N2, WORK, INFO )
```

```
LOGICAL      WANTQ
```

```
INTEGER      INFO, J1, LDQ, LDT, N, N1, N2
```

```
DOUBLE      PRECISION Q( LDQ, * ), T( LDT, * ), WORK( * )
```

PURPOSE

DLAEXC swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation.

T must be in Schur canonical form, that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

ARGUMENTS

```
WANTQ      (input) LOGICAL
            = .TRUE. : accumulate the transformation in the matrix Q;
```

= .FALSE.: do not accumulate the transformation.

N (input) INTEGER
The order of the matrix T. $N \geq 0$.

T (input/output) DOUBLE PRECISION array, dimension (LDT,N)
On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, the updated matrix T, again in Schur canonical form.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq \max(1,N)$.

Q (input/output) DOUBLE PRECISION array, dimension (LDQ,N)
On entry, if WANTQ is .TRUE., the orthogonal matrix Q. On exit, if WANTQ is .TRUE., the updated matrix Q. If WANTQ is .FALSE., Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. $LDQ \geq 1$; and if WANTQ is .TRUE., $LDQ \geq N$.

J1 (input) INTEGER
The index of the first row of the first block T11.

N1 (input) INTEGER
The order of the first block T11. $N1 = 0, 1$ or 2 .

N2 (input) INTEGER
The order of the second block T22. $N2 = 0, 1$ or 2 .

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
= 1: the transformed matrix T would be too far from Schur form; the blocks are not swapped and T and Q are unchanged.

— dlaexc.f —

```

SUBROUTINE DLAEXC( WANTQ, N, T, LDT, Q, LDQ, J1, N1, N2, WORK,
$                INFO )
*
* -- LAPACK auxiliary routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University

```

```

*      February 29, 1992
*
*      .. Scalar Arguments ..
LOGICAL          WANTQ
INTEGER          INFO, J1, LDQ, LDT, N, N1, N2
*
*      ..
*      .. Array Arguments ..
DOUBLE PRECISION Q( LDQ, * ), T( LDT, * ), WORK( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
DOUBLE PRECISION TEN
PARAMETER        ( TEN = 1.0D+1 )
INTEGER          LDD, LDX
PARAMETER        ( LDD = 4, LDX = 2 )
*
*      ..
*      .. Local Scalars ..
INTEGER          IERR, J2, J3, J4, K, ND
DOUBLE PRECISION CS, DNORM, EPS, SCALE, SMLNUM, SN, T11, T22,
$               T33, TAU, TAU1, TAU2, TEMP, THRESH, WI1, WI2,
$               WR1, WR2, XNORM
*
*      ..
*      .. Local Arrays ..
DOUBLE PRECISION D( LDD, 4 ), U( 3 ), U1( 3 ), U2( 3 ),
$               X( LDX, 2 )
*
*      ..
*      .. External Functions ..
DOUBLE PRECISION DLAMCH, DLANGE
EXTERNAL         DLAMCH, DLANGE
*
*      ..
*      .. External Subroutines ..
EXTERNAL         DLACPY, DLANV2, DLARFG, DLARFX, DLARTG, DLASY2,
$               DROT
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC        ABS, MAX
*
*      ..
*      .. Executable Statements ..
*
*      INFO = 0
*
*      Quick return if possible
*
*      IF( N.EQ.0 .OR. N1.EQ.0 .OR. N2.EQ.0 )
$      RETURN
*      IF( J1+N1.GT.N )

```



```

$   RETURN
*
*   J2 = J1 + 1
*   J3 = J1 + 2
*   J4 = J1 + 3
*
*   IF( N1.EQ.1 .AND. N2.EQ.1 ) THEN
*
*       Swap two 1-by-1 blocks.
*
*       T11 = T( J1, J1 )
*       T22 = T( J2, J2 )
*
*       Determine the transformation to perform the interchange.
*
*       CALL DLARTG( T( J1, J2 ), T22-T11, CS, SN, TEMP )
*
*       Apply transformation to the matrix T.
*
*       IF( J3.LE.N )
$           CALL DROT( N-J1-1, T( J1, J3 ), LDT, T( J2, J3 ), LDT, CS,
$                   SN )
*       CALL DROT( J1-1, T( 1, J1 ), 1, T( 1, J2 ), 1, CS, SN )
*
*       T( J1, J1 ) = T22
*       T( J2, J2 ) = T11
*
*       IF( WANTQ ) THEN
*
*           Accumulate transformation in the matrix Q.
*
*           CALL DROT( N, Q( 1, J1 ), 1, Q( 1, J2 ), 1, CS, SN )
*       END IF
*
*   ELSE
*
*       Swapping involves at least one 2-by-2 block.
*
*       Copy the diagonal block of order N1+N2 to the local array D
*       and compute its norm.
*
*       ND = N1 + N2
*       CALL DLACPY( 'Full', ND, ND, T( J1, J1 ), LDT, D, LDD )
*       DNORM = DLANGE( 'Max', ND, ND, D, LDD, WORK )
*
*       Compute machine-dependent threshold for test for accepting
*       swap.
*
*       EPS = DLAMCH( 'P' )
*       SMLNUM = DLAMCH( 'S' ) / EPS

```

```

      THRESH = MAX( TEN*EPS*DNORM, SMLNUM )
*
*      Solve T11*X - X*T22 = scale*T12 for X.
*
      CALL DLASY2( .FALSE., .FALSE., -1, N1, N2, D, LDD,
$           D( N1+1, N1+1 ), LDD, D( 1, N1+1 ), LDD, SCALE, X,
$           LDX, XNORM, IERR )
*
*      Swap the adjacent diagonal blocks.
*
      K = N1 + N1 + N2 - 3
      GO TO ( 10, 20, 30 )K
*
10    CONTINUE
*
*      N1 = 1, N2 = 2: generate elementary reflector H so that:
*
*      ( scale, X11, X12 ) H = ( 0, 0, * )
*
      U( 1 ) = SCALE
      U( 2 ) = X( 1, 1 )
      U( 3 ) = X( 1, 2 )
      CALL DLARFG( 3, U( 3 ), U, 1, TAU )
      U( 3 ) = ONE
      T11 = T( J1, J1 )
*
*      Perform swap provisionally on diagonal block in D.
*
      CALL DLARFX( 'L', 3, 3, U, TAU, D, LDD, WORK )
      CALL DLARFX( 'R', 3, 3, U, TAU, D, LDD, WORK )
*
*      Test whether to reject swap.
*
      IF( MAX( ABS( D( 3, 1 ) ), ABS( D( 3, 2 ) ), ABS( D( 3,
$           3 )-T11 ) ).GT.THRESH )GO TO 50
*
*      Accept swap: apply transformation to the entire matrix T.
*
      CALL DLARFX( 'L', 3, N-J1+1, U, TAU, T( J1, J1 ), LDT, WORK )
      CALL DLARFX( 'R', J2, 3, U, TAU, T( 1, J1 ), LDT, WORK )
*
      T( J3, J1 ) = ZERO
      T( J3, J2 ) = ZERO
      T( J3, J3 ) = T11
*
      IF( WANTQ ) THEN
*
*      Accumulate transformation in the matrix Q.
*
      CALL DLARFX( 'R', N, 3, U, TAU, Q( 1, J1 ), LDQ, WORK )

```

```

        END IF
        GO TO 40
*
20    CONTINUE
*
*    N1 = 2, N2 = 1: generate elementary reflector H so that:
*
*    H (  -X11 ) = ( * )
*       (  -X21 ) = ( 0 )
*       (  scale ) = ( 0 )
*
*    U( 1 ) = -X( 1, 1 )
*    U( 2 ) = -X( 2, 1 )
*    U( 3 ) = SCALE
*    CALL DLARFG( 3, U( 1 ), U( 2 ), 1, TAU )
*    U( 1 ) = ONE
*    T33 = T( J3, J3 )
*
*    Perform swap provisionally on diagonal block in D.
*
*    CALL DLARFX( 'L', 3, 3, U, TAU, D, LDD, WORK )
*    CALL DLARFX( 'R', 3, 3, U, TAU, D, LDD, WORK )
*
*    Test whether to reject swap.
*
*    IF( MAX( ABS( D( 2, 1 ) ), ABS( D( 3, 1 ) ), ABS( D( 1,
$      1 )-T33 ) ).GT.THRESH )GO TO 50
*
*    Accept swap: apply transformation to the entire matrix T.
*
*    CALL DLARFX( 'R', J3, 3, U, TAU, T( 1, J1 ), LDT, WORK )
*    CALL DLARFX( 'L', 3, N-J1, U, TAU, T( J1, J2 ), LDT, WORK )
*
*    T( J1, J1 ) = T33
*    T( J2, J1 ) = ZERO
*    T( J3, J1 ) = ZERO
*
*    IF( WANTQ ) THEN
*
*        Accumulate transformation in the matrix Q.
*
*        CALL DLARFX( 'R', N, 3, U, TAU, Q( 1, J1 ), LDQ, WORK )
*    END IF
*    GO TO 40
*
30    CONTINUE
*
*    N1 = 2, N2 = 2: generate elementary reflectors H(1) and H(2) so
*    that:
*

```

```

*      H(2) H(1) (  -X11  -X12 ) = (  *  *  )
*                (  -X21  -X22 )  (  0  *  )
*                (  scale    0  )  (  0  0  )
*                (    0  scale )  (  0  0  )
*
      U1( 1 ) = -X( 1, 1 )
      U1( 2 ) = -X( 2, 1 )
      U1( 3 ) = SCALE
      CALL DLARFG( 3, U1( 1 ), U1( 2 ), 1, TAU1 )
      U1( 1 ) = ONE
*
      TEMP = -TAU1*( X( 1, 2 )+U1( 2 )*X( 2, 2 ) )
      U2( 1 ) = -TEMP*U1( 2 ) - X( 2, 2 )
      U2( 2 ) = -TEMP*U1( 3 )
      U2( 3 ) = SCALE
      CALL DLARFG( 3, U2( 1 ), U2( 2 ), 1, TAU2 )
      U2( 1 ) = ONE
*
*      Perform swap provisionally on diagonal block in D.
*
      CALL DLARFX( 'L', 3, 4, U1, TAU1, D, LDD, WORK )
      CALL DLARFX( 'R', 4, 3, U1, TAU1, D, LDD, WORK )
      CALL DLARFX( 'L', 3, 4, U2, TAU2, D( 2, 1 ), LDD, WORK )
      CALL DLARFX( 'R', 4, 3, U2, TAU2, D( 1, 2 ), LDD, WORK )
*
*      Test whether to reject swap.
*
      IF( MAX( ABS( D( 3, 1 ) ), ABS( D( 3, 2 ) ), ABS( D( 4, 1 ) ),
$      ABS( D( 4, 2 ) ) ).GT.THRESH )GO TO 50
*
*      Accept swap: apply transformation to the entire matrix T.
*
      CALL DLARFX( 'L', 3, N-J1+1, U1, TAU1, T( J1, J1 ), LDT, WORK )
      CALL DLARFX( 'R', J4, 3, U1, TAU1, T( 1, J1 ), LDT, WORK )
      CALL DLARFX( 'L', 3, N-J1+1, U2, TAU2, T( J2, J1 ), LDT, WORK )
      CALL DLARFX( 'R', J4, 3, U2, TAU2, T( 1, J2 ), LDT, WORK )
*
      T( J3, J1 ) = ZERO
      T( J3, J2 ) = ZERO
      T( J4, J1 ) = ZERO
      T( J4, J2 ) = ZERO
*
      IF( WANTQ ) THEN
*
*      Accumulate transformation in the matrix Q.
*
      CALL DLARFX( 'R', N, 3, U1, TAU1, Q( 1, J1 ), LDQ, WORK )
      CALL DLARFX( 'R', N, 3, U2, TAU2, Q( 1, J2 ), LDQ, WORK )
      END IF
*

```

```

40    CONTINUE
*
      IF( N2.EQ.2 ) THEN
*
*        Standardize new 2-by-2 block T11
*
      CALL DLANV2( T( J1, J1 ), T( J1, J2 ), T( J2, J1 ),
$              T( J2, J2 ), WR1, WI1, WR2, WI2, CS, SN )
      CALL DROT( N-J1-1, T( J1, J1+2 ), LDT, T( J2, J1+2 ), LDT,
$              CS, SN )
      CALL DROT( J1-1, T( 1, J1 ), 1, T( 1, J2 ), 1, CS, SN )
      IF( WANTQ )
$          CALL DROT( N, Q( 1, J1 ), 1, Q( 1, J2 ), 1, CS, SN )
      END IF
*
      IF( N1.EQ.2 ) THEN
*
*        Standardize new 2-by-2 block T22
*
      J3 = J1 + N2
      J4 = J3 + 1
      CALL DLANV2( T( J3, J3 ), T( J3, J4 ), T( J4, J3 ),
$              T( J4, J4 ), WR1, WI1, WR2, WI2, CS, SN )
      IF( J3+2.LE.N )
$          CALL DROT( N-J3-1, T( J3, J3+2 ), LDT, T( J4, J3+2 ),
$              LDT, CS, SN )
      CALL DROT( J3-1, T( 1, J3 ), 1, T( 1, J4 ), 1, CS, SN )
      IF( WANTQ )
$          CALL DROT( N, Q( 1, J3 ), 1, Q( 1, J4 ), 1, CS, SN )
      END IF
*
      END IF
      RETURN
*
      Exit with INFO = 1 if swap was rejected.
*
50 CONTINUE
   INFO = 1
   RETURN
*
*   End of DLAEXC
*
END

```

```

(let* ((zero 0.0) (one 1.0) (ten 10.0) (ldd 4) (ldx 2))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 10.0 10.0) ten)
            (type (fixnum 4 4) ldd)
            (type (fixnum 2 2) ldx))
  (defun dlaexc (wantq n t$ ldt q ldq j1 n1 n2 work info)
    (declare (type (simple-array double-float (*)) work q t$)
              (type fixnum info n2 n1 j1 ldq ldt n)
              (type (member t nil) wantq))
    (f2cl-lib:with-multi-array-data
      ((t$ double-float t$-%data% t$-%offset%)
       (q double-float q-%data% q-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((d
               (make-array (the fixnum (reduce #'* (list ldd 4)))
                           :element-type 'double-float))
              (u (make-array 3 :element-type 'double-float))
              (u1 (make-array 3 :element-type 'double-float))
              (u2 (make-array 3 :element-type 'double-float))
              (x
               (make-array (the fixnum (reduce #'* (list ldx 2)))
                           :element-type 'double-float))
              (cs 0.0) (dnorm 0.0) (eps 0.0) (scale 0.0) (smlnum 0.0) (sn 0.0)
              (t11 0.0) (t22 0.0) (t33 0.0) (tau 0.0) (tau1 0.0) (tau2 0.0)
              (temp 0.0) (thresh 0.0) (wi1 0.0) (wi2 0.0) (wr1 0.0) (wr2 0.0)
              (xnorm 0.0) (ierr 0) (j2 0) (j3 0) (j4 0) (k 0) (nd 0))
        (declare (type (simple-array double-float (3)) u u1 u2)
                  (type (simple-array double-float (*)) d x)
                  (type (double-float) cs dnorm eps scale smlnum sn t11 t22 t33
                          tau tau1 tau2 temp thresh wi1 wi2 wr1 wr2
                          xnorm)
                  (type fixnum ierr j2 j3 j4 k nd))
        (setf info 0)
        (if (or (= n 0) (= n1 0) (= n2 0)) (go end_label))
        (if (> (f2cl-lib:int-add j1 n1) n) (go end_label))
        (setf j2 (f2cl-lib:int-add j1 1))
        (setf j3 (f2cl-lib:int-add j1 2))
        (setf j4 (f2cl-lib:int-add j1 3))
        (cond
          ((and (= n1 1) (= n2 1))
           (setf t11
                 (f2cl-lib:fref t$-%data%
                               (j1 j1)
                               ((1 ldt) (1 *)))
                 t$-%offset%))
           (setf t22
                 (f2cl-lib:fref t$-%data%
                               (j2 j2)
                               ((1 ldt) (1 *)))

```

```

                                t$-%offset%))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dclartg
    (f2cl-lib:fref t$-%data% (j1 j2) ((1 ldt) (1 *)) t$-%offset%)
    (- t22 t11) cs sn temp)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf temp var-4))
(if (<= j3 n)
  (drot (f2cl-lib:int-sub n j1 1)
    (f2cl-lib:array-slice t$ double-float (j1 j3) ((1 ldt) (1 *)))
    ldt
    (f2cl-lib:array-slice t$ double-float (j2 j3) ((1 ldt) (1 *)))
    ldt cs sn))
(drot (f2cl-lib:int-sub j1 1)
  (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) 1
  (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *))) 1 cs
  sn)
(setf (f2cl-lib:fref t$-%data% (j1 j1) ((1 ldt) (1 *)) t$-%offset%)
  t22)
(setf (f2cl-lib:fref t$-%data% (j2 j2) ((1 ldt) (1 *)) t$-%offset%)
  t11)
(cond
  (wantq
    (drot n
      (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *))) 1
      (f2cl-lib:array-slice q double-float (1 j2) ((1 ldq) (1 *))) 1
      cs sn))))
(t
  (tagbody
    (setf nd (f2cl-lib:int-add n1 n2))
    (dlacpy "Full" nd nd
      (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
      ldt d ldd)
    (setf dnorm (dlange "Max" nd nd d ldd work))
    (setf eps (dlamch "P"))
    (setf smlnum (/ (dlamch "S") eps))
    (setf thresh (max (* ten eps dnorm) smlnum))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15)
      (dlasy2 nil nil -1 n1 n2 d ldd
        (f2cl-lib:array-slice d
          double-float
          ((+ n1 1) (f2cl-lib:int-add n1 1))
          ((1 ldd) (1 4)))
        ldd
        (f2cl-lib:array-slice d
          double-float

```

```

(1 (f2cl-lib:int-add n1 1))
((1 ldd) (1 4)))
  ldd scale x ldx xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-12 var-13))
(setf scale var-11)
(setf xnorm var-14)
(setf ierr var-15))
(setf k (f2cl-lib:int-sub (f2cl-lib:int-add n1 n1 n2) 3))
(f2cl-lib:computed-goto (label10 label20 label30) k)
label10
(setf (f2cl-lib:fref u (1) ((1 3))) scale)
(setf (f2cl-lib:fref u (2) ((1 3)))
  (f2cl-lib:fref x (1 1) ((1 ldx) (1 2))))
(setf (f2cl-lib:fref u (3) ((1 3)))
  (f2cl-lib:fref x (1 2) ((1 ldx) (1 2))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlarf3g 3 (f2cl-lib:fref u (3) ((1 3))) u 1 tau)
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref u (3) ((1 3))) var-1)
  (setf tau var-4))
(setf (f2cl-lib:fref u (3) ((1 3))) one)
(setf t11
  (f2cl-lib:fref t$-%data%
    (j1 j1)
    ((1 ldt) (1 *))
    t$-%offset%))
(dlarfx "L" 3 3 u tau d ldd work)
(dlarfx "R" 3 3 u tau d ldd work)
(if
  (>
    (max (abs (f2cl-lib:fref d (3 1) ((1 ldd) (1 4)))
      (abs (f2cl-lib:fref d (3 2) ((1 ldd) (1 4)))
      (abs (- (f2cl-lib:fref d (3 3) ((1 ldd) (1 4))) t11)))
    thresh)
  (go label50))
(dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u tau
  (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
  ldt work)
(dlarfx "R" j2 3 u tau
  (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) ldt
  work)
(setf (f2cl-lib:fref t$-%data%
  (j3 j1)
  ((1 ldt) (1 *))
  t$-%offset%)
  zero)
(setf (f2cl-lib:fref t$-%data%
  (j3 j2)
  ((1 ldt) (1 *)))

```



```

        t33)
(setf (f2cl-lib:fref t$-%data%
                    (j2 j1)
                    ((1 ldt) (1 *)))
      t$-%offset%)

      zero)
(setf (f2cl-lib:fref t$-%data%
                    (j3 j1)
                    ((1 ldt) (1 *)))
      t$-%offset%)

      zero)
(cond
  (wantq
    (dlarfx "R" n 3 u tau
      (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *)))
      ldq work)))
(go label40)
label30
(setf (f2cl-lib:fref u1 (1) ((1 3)))
      (- (f2cl-lib:fref x (1 1) ((1 ldx) (1 2)))))
(setf (f2cl-lib:fref u1 (2) ((1 3)))
      (- (f2cl-lib:fref x (2 1) ((1 ldx) (1 2)))))
(setf (f2cl-lib:fref u1 (3) ((1 3))) scale)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlarf3g 3 (f2cl-lib:fref u1 (1) ((1 3)))
    (f2cl-lib:array-slice u1 double-float (2) ((1 3))) 1 tau1)
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref u1 (1) ((1 3))) var-1)
  (setf tau1 var-4))
(setf (f2cl-lib:fref u1 (1) ((1 3))) one)
(setf temp
  (* (- tau1)
    (+ (f2cl-lib:fref x (1 2) ((1 ldx) (1 2)))
      (* (f2cl-lib:fref u1 (2) ((1 3)))
        (f2cl-lib:fref x (2 2) ((1 ldx) (1 2)))))))
(setf (f2cl-lib:fref u2 (1) ((1 3)))
      (- (* (- temp) (f2cl-lib:fref u1 (2) ((1 3)))
        (f2cl-lib:fref x (2 2) ((1 ldx) (1 2)))))
(setf (f2cl-lib:fref u2 (2) ((1 3)))
      (* (- temp) (f2cl-lib:fref u1 (3) ((1 3)))))
(setf (f2cl-lib:fref u2 (3) ((1 3))) scale)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlarf3g 3 (f2cl-lib:fref u2 (1) ((1 3)))
    (f2cl-lib:array-slice u2 double-float (2) ((1 3))) 1 tau2)
  (declare (ignore var-0 var-2 var-3))
  (setf (f2cl-lib:fref u2 (1) ((1 3))) var-1)
  (setf tau2 var-4))
(setf (f2cl-lib:fref u2 (1) ((1 3))) one)
(dlarfx "L" 3 4 u1 tau1 d ldd work)
(dlarfx "R" 4 3 u1 tau1 d ldd work)

```

```

(dlarfx "L" 3 4 u2 tau2
  (f2cl-lib:array-slice d double-float (2 1) ((1 ldd) (1 4))) ldd
  work)
(dlarfx "R" 4 3 u2 tau2
  (f2cl-lib:array-slice d double-float (1 2) ((1 ldd) (1 4))) ldd
  work)
(if
  (>
    (max (abs (f2cl-lib:fref d (3 1) ((1 ldd) (1 4))))
      (abs (f2cl-lib:fref d (3 2) ((1 ldd) (1 4))))
      (abs (f2cl-lib:fref d (4 1) ((1 ldd) (1 4))))
      (abs (f2cl-lib:fref d (4 2) ((1 ldd) (1 4)))))
    thresh)
  (go label50))
(dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u1 tau1
  (f2cl-lib:array-slice t$ double-float (j1 j1) ((1 ldt) (1 *)))
  ldt work)
(dlarfx "R" j4 3 u1 tau1
  (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *))) ldt
  work)
(dlarfx "L" 3 (f2cl-lib:int-add (f2cl-lib:int-sub n j1) 1) u2 tau2
  (f2cl-lib:array-slice t$ double-float (j2 j1) ((1 ldt) (1 *)))
  ldt work)
(dlarfx "R" j4 3 u2 tau2
  (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *))) ldt
  work)
(setf (f2cl-lib:fref t$-%data%
                  (j3 j1)
                  ((1 ldt) (1 *)))
      t$-%offset%)
  zero)
(setf (f2cl-lib:fref t$-%data%
                  (j3 j2)
                  ((1 ldt) (1 *)))
      t$-%offset%)
  zero)
(setf (f2cl-lib:fref t$-%data%
                  (j4 j1)
                  ((1 ldt) (1 *)))
      t$-%offset%)
  zero)
(setf (f2cl-lib:fref t$-%data%
                  (j4 j2)
                  ((1 ldt) (1 *)))
      t$-%offset%)
  zero)
(cond
  (wantq
    (dlarfx "R" n 3 u1 tau1
      (f2cl-lib:array-slice q double-float (1 j1) ((1 ldq) (1 *)))

```

```

      ldq work)
      (dlarfz "R" n 3 u2 tau2
        (f2cl-lib:array-slice q double-float (1 j2) ((1 ldq) (1 *)))
        ldq work)))
label40
(cond
  ((= n2 2)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dlanv2
        (f2cl-lib:fref t$-%data%
          (j1 j1)
          ((1 ldt) (1 *))
          t$-%offset%)
        (f2cl-lib:fref t$-%data%
          (j1 j2)
          ((1 ldt) (1 *))
          t$-%offset%)
        (f2cl-lib:fref t$-%data%
          (j2 j1)
          ((1 ldt) (1 *))
          t$-%offset%)
        (f2cl-lib:fref t$-%data%
          (j2 j2)
          ((1 ldt) (1 *))
          t$-%offset%)
        wr1 wi1 wr2 wi2 cs sn)
      (declare (ignore))
      (setf (f2cl-lib:fref t$-%data%
        (j1 j1)
        ((1 ldt) (1 *))
        t$-%offset%)
        var-0)
      (setf (f2cl-lib:fref t$-%data%
        (j1 j2)
        ((1 ldt) (1 *))
        t$-%offset%)
        var-1)
      (setf (f2cl-lib:fref t$-%data%
        (j2 j1)
        ((1 ldt) (1 *))
        t$-%offset%)
        var-2)
      (setf (f2cl-lib:fref t$-%data%
        (j2 j2)
        ((1 ldt) (1 *))
        t$-%offset%)
        var-3)
      (setf wr1 var-4)

```

```

      (setf wi1 var-5)
      (setf wr2 var-6)
      (setf wi2 var-7)
      (setf cs var-8)
      (setf sn var-9))
(drot (f2cl-lib:int-sub n j1 1)
      (f2cl-lib:array-slice t$
                             double-float
                             (j1 (f2cl-lib:int-add j1 2))
                             ((1 ldt) (1 *))))

ldt
(f2cl-lib:array-slice t$
                     double-float
                     (j2 (f2cl-lib:int-add j1 2))
                     ((1 ldt) (1 *)))

ldt cs sn)
(drot (f2cl-lib:int-sub j1 1)
      (f2cl-lib:array-slice t$ double-float (1 j1) ((1 ldt) (1 *)))
      1
      (f2cl-lib:array-slice t$ double-float (1 j2) ((1 ldt) (1 *)))
      1 cs sn)
(if wantq
    (drot n
          (f2cl-lib:array-slice q
                                double-float
                                (1 j1)
                                ((1 ldq) (1 *)))
          1
          (f2cl-lib:array-slice q
                                double-float
                                (1 j2)
                                ((1 ldq) (1 *)))
          1 cs sn))))
(cond
  ((= n1 2)
   (setf j3 (f2cl-lib:int-add j1 n2))
   (setf j4 (f2cl-lib:int-add j3 1))
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9)
     (dlanv2
      (f2cl-lib:fref t$-%data%
                     (j3 j3)
                     ((1 ldt) (1 *))
                     t$-%offset%)
      (f2cl-lib:fref t$-%data%
                     (j3 j4)
                     ((1 ldt) (1 *))
                     t$-%offset%)
      (f2cl-lib:fref t$-%data%

```

```

                (j4 j3)
                ((1 ldt) (1 *))
                t$-%offset%)
(f2cl-lib:fref t$-%data%
                (j4 j4)
                ((1 ldt) (1 *))
                t$-%offset%)
  wr1 wi1 wr2 wi2 cs sn)
(declare (ignore))
(setf (f2cl-lib:fref t$-%data%
                    (j3 j3)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      var-0)
(setf (f2cl-lib:fref t$-%data%
                    (j3 j4)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      var-1)
(setf (f2cl-lib:fref t$-%data%
                    (j4 j3)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      var-2)
(setf (f2cl-lib:fref t$-%data%
                    (j4 j4)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      var-3)
(setf wr1 var-4)
(setf wi1 var-5)
(setf wr2 var-6)
(setf wi2 var-7)
(setf cs var-8)
(setf sn var-9))
(if (<= (f2cl-lib:int-add j3 2) n)
    (drot (f2cl-lib:int-sub n j3 1)
          (f2cl-lib:array-slice t$
                                double-float
                                (j3 (f2cl-lib:int-add j3 2))
                                ((1 ldt) (1 *)))

          ldt
          (f2cl-lib:array-slice t$
                                double-float
                                (j4 (f2cl-lib:int-add j3 2))
                                ((1 ldt) (1 *)))

          ldt cs sn))
(drot (f2cl-lib:int-sub j3 1)
      (f2cl-lib:array-slice t$ double-float (1 j3) ((1 ldt) (1 *)))
      1

```

```

        (f2cl-lib:array-slice t$ double-float (1 j4) ((1 ldt) (1 *)))
        1 cs sn)
      (if wantq
        (drot n
          (f2cl-lib:array-slice q
            double-float
            (1 j3)
            ((1 ldq) (1 *)))
          1
          (f2cl-lib:array-slice q
            double-float
            (1 j4)
            ((1 ldq) (1 *)))
          1 cs sn))))))
      (go end_label)
label50
      (setf info 1)
end_label
      (return (values nil nil nil nil nil nil nil nil nil nil info))))))

```

dlahqr LAPACK

— dlahqr.input —

```

)set break resume
)sys rm -f dlahqr.output
)spool dlahqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlahqr.help —

```

=====
dlahqr examples
=====
=====

```

Man Page Details

=====

NAME

DLAHQR - An auxiliary routine called by DHSEQR to update the eigenvalues and Schur decomposition already computed by DHSEQR, by dealing with the Hessenberg submatrix in rows and columns ILO to IHI

SYNOPSIS

```
SUBROUTINE DLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI, ILOZ,
                  IHIZ, Z, LDZ, INFO )
```

```
      INTEGER      IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, N
```

```
      LOGICAL      WANTT, WANTZ
```

```
      DOUBLE      PRECISION H( LDH, * ), WI( * ), WR( * ), Z( LDZ, * )
```

Purpose

=====

DLAHQR is an auxiliary routine called by DHSEQR to update the eigenvalues and Schur decomposition already computed by DHSEQR, by dealing with the Hessenberg submatrix in rows and columns ILO to IHI.

Arguments

=====

WANTT (input) LOGICAL
 = .TRUE. : the full Schur form T is required;
 = .FALSE.: only eigenvalues are required.

WANTZ (input) LOGICAL
 = .TRUE. : the matrix of Schur vectors Z is required;
 = .FALSE.: Schur vectors are not required.

N (input) INTEGER
 The order of the matrix H. N >= 0.

ILO (input) INTEGER
 IHI (input) INTEGER
 It is assumed that H is already upper quasi-triangular in rows and columns IHI+1:N, and that H(ILO,ILO-1) = 0 (unless ILO = 1). DLAHQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE..
 1 <= ILO <= max(1,IHI); IHI <= N.

H (input/output) DOUBLE PRECISION array, dimension (LDH,N)
 On entry, the upper Hessenberg matrix H.

On exit, if WANTT is `.TRUE.`, H is upper quasi-triangular in rows and columns ILO:IHI, with any 2-by-2 diagonal blocks in standard form. If WANTT is `.FALSE.`, the contents of H are unspecified on exit.

LDH (input) INTEGER
The leading dimension of the array H. LDH \geq max(1,N).

WR (output) DOUBLE PRECISION array, dimension (N)
WI (output) DOUBLE PRECISION array, dimension (N)
The real and imaginary parts, respectively, of the computed eigenvalues ILO to IHI are stored in the corresponding elements of WR and WI. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)th, with $WI(i) > 0$ and $WI(i+1) < 0$. If WANTT is `.TRUE.`, the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with $WR(i) = H(i,i)$, and, if $H(i:i+1,i:i+1)$ is a 2-by-2 diagonal block, $WI(i) = \sqrt{H(i+1,i)*H(i,i+1)}$ and $WI(i+1) = -WI(i)$.

ILOZ (input) INTEGER
IHIZ (input) INTEGER
Specify the rows of Z to which transformations must be applied if WANTZ is `.TRUE.`.
 $1 \leq ILOZ \leq ILO$; $IHI \leq IHIZ \leq N$.

Z (input/output) DOUBLE PRECISION array, dimension (LDZ,N)
If WANTZ is `.TRUE.`, on entry Z must contain the current matrix Z of transformations accumulated by DHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI).
If WANTZ is `.FALSE.`, Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ \geq max(1,N).

INFO (output) INTEGER
= 0: successful exit
> 0: DLAHQR failed to compute all the eigenvalues ILO to IHI in a total of $30*(IHI-ILO+1)$ iterations; if INFO = i, elements i+1:ihi of WR and WI contain those eigenvalues which have been successfully computed.

Further Details
=====

2-96 Based on modifications by
David Day, Sandia National Laboratory, USA

— dlahqr.f —

```

SUBROUTINE DLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, WR, WI,
$                ILOZ, IHIZ, Z, LDZ, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    June 30, 1999
*
*    .. Scalar Arguments ..
LOGICAL          WANTT, WANTZ
INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, N
*
*    ..
*    .. Array Arguments ..
DOUBLE PRECISION H( LDH, * ), WI( * ), WR( * ), Z( LDZ, * )
*
*    ..
*
*    =====
*
*    .. Parameters ..
DOUBLE PRECISION ZERO, ONE, HALF
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0, HALF = 0.5D0 )
DOUBLE PRECISION DAT1, DAT2
PARAMETER        ( DAT1 = 0.75D+0, DAT2 = -0.4375D+0 )
*
*    ..
*    .. Local Scalars ..
INTEGER          I, I1, I2, ITN, ITS, J, K, L, M, NH, NR, NZ
DOUBLE PRECISION AVE, CS, DISC, H00, H10, H11, H12, H21, H22,
$               H33, H33S, H43H34, H44, H44S, OVFL, S, SMLNUM,
$               SN, SUM, T1, T2, T3, TST1, ULP, UNFL, V1, V2,
$               V3
*
*    ..
*    .. Local Arrays ..
DOUBLE PRECISION V( 3 ), WORK( 1 )
*
*    ..
*    .. External Functions ..
DOUBLE PRECISION DLAMCH, DLANHS
EXTERNAL          DLAMCH, DLANHS
*
*    ..
*    .. External Subroutines ..
EXTERNAL          DCOPY, DLANV2, DLARFG, DROT
*
*    ..
*    .. Intrinsic Functions ..
INTRINSIC         ABS, MAX, MIN, SIGN, SQRT
*
*    ..
*    .. Executable Statements ..

```

```

*
      INFO = 0
*
*   Quick return if possible
*
      IF( N.EQ.0 )
$      RETURN
      IF( ILO.EQ.IHI ) THEN
          WR( ILO ) = H( ILO, ILO )
          WI( ILO ) = ZERO
          RETURN
      END IF
*
      NH = IHI - ILO + 1
      NZ = IHIZ - ILOZ + 1
*
*   Set machine-dependent constants for the stopping criterion.
*   If norm(H) <= sqrt(OVFL), overflow should not occur.
*
      UNFL = DLAMCH( 'Safe minimum' )
      OVFL = ONE / UNFL
      CALL DLABAD( UNFL, OVFL )
      ULP = DLAMCH( 'Precision' )
      SMLNUM = UNFL*( NH / ULP )
*
*   I1 and I2 are the indices of the first row and last column of H
*   to which transformations must be applied. If eigenvalues only are
*   being computed, I1 and I2 are set inside the main loop.
*
      IF( WANTT ) THEN
          I1 = 1
          I2 = N
      END IF
*
*   ITN is the total number of QR iterations allowed.
*
      ITN = 30*NH
*
*   The main loop begins here. I is the loop index and decreases from
*   IHI to ILO in steps of 1 or 2. Each iteration of the loop works
*   with the active submatrix in rows and columns L to I.
*   Eigenvalues I+1 to IHI have already converged. Either L = ILO or
*   H(L,L-1) is negligible so that the matrix splits.
*
      I = IHI
10  CONTINUE
      L = ILO
      IF( I.LT.ILO )
$      GO TO 150
*

```

```

*      Perform QR iterations on rows and columns ILO to I until a
*      submatrix of order 1 or 2 splits off at the bottom because a
*      subdiagonal element has become negligible.
*
      DO 130 ITS = 0, ITN
*
*      Look for a single small subdiagonal element.
*
      DO 20 K = I, L + 1, -1
        TST1 = ABS( H( K-1, K-1 ) ) + ABS( H( K, K ) )
        IF( TST1.EQ.ZERO )
          $      TST1 = DLANHS( '1', I-L+1, H( L, L ), LDH, WORK )
          IF( ABS( H( K, K-1 ) ) .LE. MAX( ULP*TST1, SMLNUM ) )
            $      GO TO 30
20      CONTINUE
30      CONTINUE
        L = K
        IF( L.GT.ILO ) THEN
*
*      H(L,L-1) is negligible
*
          H( L, L-1 ) = ZERO
        END IF
*
*      Exit from loop if a submatrix of order 1 or 2 has split off.
*
        IF( L.GE.I-1 )
          $      GO TO 140
*
*      Now the active submatrix is in rows and columns L to I. If
*      eigenvalues only are being computed, only the active submatrix
*      need be transformed.
*
        IF( .NOT.WANTT ) THEN
          I1 = L
          I2 = I
        END IF
*
        IF( ITS.EQ.10 .OR. ITS.EQ.20 ) THEN
*
*      Exceptional shift.
*
          S = ABS( H( I, I-1 ) ) + ABS( H( I-1, I-2 ) )
          H44 = DAT1*S + H( I, I )
          H33 = H44
          H43H34 = DAT2*S*S
        ELSE
*
*      Prepare to use Francis' double shift
*      (i.e. 2nd degree generalized Rayleigh quotient)

```

```

*
      H44 = H( I, I )
      H33 = H( I-1, I-1 )
      H43H34 = H( I, I-1 )*H( I-1, I )
      S = H( I-1, I-2 )*H( I-1, I-2 )
      DISC = ( H33-H44 )*HALF
      DISC = DISC*DISC + H43H34
      IF( DISC.GT.ZERO ) THEN

*
*       Real roots: use Wilkinson's shift twice
*
      DISC = SQRT( DISC )
      AVE = HALF*( H33+H44 )
      IF( ABS( H33 )-ABS( H44 ).GT.ZERO ) THEN
        H33 = H33*H44 - H43H34
        H44 = H33 / ( SIGN( DISC, AVE )+AVE )
      ELSE
        H44 = SIGN( DISC, AVE ) + AVE
      END IF
      H33 = H44
      H43H34 = ZERO
    END IF
  END IF

*
*       Look for two consecutive small subdiagonal elements.
*
DO 40 M = I - 2, L, -1
  *       Determine the effect of starting the double-shift QR
  *       iteration at row M, and see if this would make H(M,M-1)
  *       negligible.
  *
      H11 = H( M, M )
      H22 = H( M+1, M+1 )
      H21 = H( M+1, M )
      H12 = H( M, M+1 )
      H44S = H44 - H11
      H33S = H33 - H11
      V1 = ( H33S*H44S-H43H34 ) / H21 + H12
      V2 = H22 - H11 - H33S - H44S
      V3 = H( M+2, M+1 )
      S = ABS( V1 ) + ABS( V2 ) + ABS( V3 )
      V1 = V1 / S
      V2 = V2 / S
      V3 = V3 / S
      V( 1 ) = V1
      V( 2 ) = V2
      V( 3 ) = V3
      IF( M.EQ.L )
$        GO TO 50
      H00 = H( M-1, M-1 )

```

```

      H10 = H( M, M-1 )
      TST1 = ABS( V1 )*( ABS( H00 )+ABS( H11 )+ABS( H22 ) )
      IF( ABS( H10 )*( ABS( V2 )+ABS( V3 ) ).LE.ULP*TST1 )
        $      GO TO 50
40    CONTINUE
50    CONTINUE
*
*      Double-shift QR step
*
      DO 120 K = M, I - 1
*
*      The first iteration of this loop determines a reflection G
*      from the vector V and applies it from left and right to H,
*      thus creating a nonzero bulge below the subdiagonal.
*
*      Each subsequent iteration determines a reflection G to
*      restore the Hessenberg form in the (K-1)th column, and thus
*      chases the bulge one step toward the bottom of the active
*      submatrix. NR is the order of G.
*
      NR = MIN( 3, I-K+1 )
      IF( K.GT.M )
        $      CALL DCOPY( NR, H( K, K-1 ), 1, V, 1 )
      CALL DLARFG( NR, V( 1 ), V( 2 ), 1, T1 )
      IF( K.GT.M ) THEN
        H( K, K-1 ) = V( 1 )
        H( K+1, K-1 ) = ZERO
        IF( K.LT.I-1 )
          $      H( K+2, K-1 ) = ZERO
      ELSE IF( M.GT.L ) THEN
        H( K, K-1 ) = -H( K, K-1 )
      END IF
      V2 = V( 2 )
      T2 = T1*V2
      IF( NR.EQ.3 ) THEN
        V3 = V( 3 )
        T3 = T1*V3
*
*      Apply G from the left to transform the rows of the matrix
*      in columns K to I2.
*
      DO 60 J = K, I2
        SUM = H( K, J ) + V2*H( K+1, J ) + V3*H( K+2, J )
        H( K, J ) = H( K, J ) - SUM*T1
        H( K+1, J ) = H( K+1, J ) - SUM*T2
        H( K+2, J ) = H( K+2, J ) - SUM*T3
60    CONTINUE
*
*      Apply G from the right to transform the columns of the
*      matrix in rows I1 to min(K+3,I).

```

```

*
      DO 70 J = I1, MIN( K+3, I )
        SUM = H( J, K ) + V2*H( J, K+1 ) + V3*H( J, K+2 )
        H( J, K ) = H( J, K ) - SUM*T1
        H( J, K+1 ) = H( J, K+1 ) - SUM*T2
        H( J, K+2 ) = H( J, K+2 ) - SUM*T3
70      CONTINUE
*
      IF( WANTZ ) THEN
*
*        Accumulate transformations in the matrix Z
*
      DO 80 J = ILOZ, IHIZ
        SUM = Z( J, K ) + V2*Z( J, K+1 ) + V3*Z( J, K+2 )
        Z( J, K ) = Z( J, K ) - SUM*T1
        Z( J, K+1 ) = Z( J, K+1 ) - SUM*T2
        Z( J, K+2 ) = Z( J, K+2 ) - SUM*T3
80      CONTINUE
      END IF
      ELSE IF( NR.EQ.2 ) THEN
*
*        Apply G from the left to transform the rows of the matrix
*        in columns K to I2.
*
      DO 90 J = K, I2
        SUM = H( K, J ) + V2*H( K+1, J )
        H( K, J ) = H( K, J ) - SUM*T1
        H( K+1, J ) = H( K+1, J ) - SUM*T2
90      CONTINUE
*
*        Apply G from the right to transform the columns of the
*        matrix in rows I1 to min(K+3,I).
*
      DO 100 J = I1, I
        SUM = H( J, K ) + V2*H( J, K+1 )
        H( J, K ) = H( J, K ) - SUM*T1
        H( J, K+1 ) = H( J, K+1 ) - SUM*T2
100     CONTINUE
*
      IF( WANTZ ) THEN
*
*        Accumulate transformations in the matrix Z
*
      DO 110 J = ILOZ, IHIZ
        SUM = Z( J, K ) + V2*Z( J, K+1 )
        Z( J, K ) = Z( J, K ) - SUM*T1
        Z( J, K+1 ) = Z( J, K+1 ) - SUM*T2
110     CONTINUE
      END IF
    END IF

```

```

120    CONTINUE
*
130    CONTINUE
*
*      Failure to converge in remaining number of iterations
*
      INFO = I
      RETURN
*
140    CONTINUE
*
      IF( L.EQ.I ) THEN
*
*      H(I,I-1) is negligible: one eigenvalue has converged.
*
      WR( I ) = H( I, I )
      WI( I ) = ZERO
      ELSE IF( L.EQ.I-1 ) THEN
*
*      H(I-1,I-2) is negligible: a pair of eigenvalues have converged.
*
*      Transform the 2-by-2 submatrix to standard Schur form,
*      and compute and store the eigenvalues.
*
      CALL DLANV2( H( I-1, I-1 ), H( I-1, I ), H( I, I-1 ),
$                H( I, I ), WR( I-1 ), WI( I-1 ), WR( I ), WI( I ),
$                CS, SN )
*
      IF( WANTT ) THEN
*
*      Apply the transformation to the rest of H.
*
      IF( I2.GT.I )
$        CALL DROT( I2-I, H( I-1, I+1 ), LDH, H( I, I+1 ), LDH,
$                  CS, SN )
      CALL DROT( I-I1-1, H( I1, I-1 ), 1, H( I1, I ), 1, CS, SN )
      END IF
      IF( WANTZ ) THEN
*
*      Apply the transformation to Z.
*
      CALL DROT( NZ, Z( ILOZ, I-1 ), 1, Z( ILOZ, I ), 1, CS, SN )
      END IF
      END IF
*
*      Decrement number of remaining iterations, and return to start of
*      the main loop with new value of I.
*
      ITN = ITN - ITS
      I = L - 1

```



```

      GO TO 10
*
150 CONTINUE
      RETURN
*
*      End of DLAHQQR
*
      END

```

— LAPACK dlahqr —

```

(let* ((zero 0.0) (one 1.0) (half 0.5) (dat1 0.75) (dat2 (- 0.4375)))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 0.5 0.5) half)
            (type (double-float 0.75 0.75) dat1)
            (type (double-float) dat2))
  (defun dlahqr (wantt wantz n ilo ihi h ldh wr wi iloz ihiz z ldz info)
    (declare (type (simple-array double-float (*)) z wi wr h)
              (type fixnum info ldz ihiz iloz ldh ihi ilo n)
              (type (member t nil) wantz wantt))
    (f2cl-lib:with-multi-array-data
      ((h double-float h-%data% h-%offset%)
       (wr double-float wr-%data% wr-%offset%)
       (wi double-float wi-%data% wi-%offset%)
       (z double-float z-%data% z-%offset%))
      (prog ((v (make-array 3 :element-type 'double-float))
              (work (make-array 1 :element-type 'double-float)) (ave 0.0)
              (cs 0.0) (disc 0.0) (h00 0.0) (h10 0.0) (h11 0.0) (h12 0.0)
              (h21 0.0) (h22 0.0) (h33 0.0) (h33s 0.0) (h43h34 0.0) (h44 0.0)
              (h44s 0.0) (ovfl 0.0) (s 0.0) (smlnum 0.0) (sn 0.0) (sum 0.0)
              (t1 0.0) (t2 0.0) (t3 0.0) (tst1 0.0) (ulp 0.0) (unfl 0.0)
              (v1 0.0) (v2 0.0) (v3 0.0) (i 0) (i1 0) (i2 0) (itn 0) (its 0)
              (j 0) (k 0) (l 0) (m 0) (nh 0) (nr 0) (nz 0))
        (declare (type (simple-array double-float (3)) v)
                  (type (simple-array double-float (1)) work)
                  (type (double-float) ave cs disc h00 h10 h11 h12 h21 h22 h33
                        h33s h43h34 h44 h44s ovfl s smlnum sn sum
                        t1 t2 t3 tst1 ulp unfl v1 v2 v3)
                  (type fixnum i i1 i2 itn its j k l m nh nr nz))
        (setf info 0)
        (if (= n 0) (go end_label))
        (cond
         ((= ilo ihi)
          (setf (f2cl-lib:fref wr-%data% (ilo) ((1 *)) wr-%offset%)
                (f2cl-lib:fref h-%data%

```

```

                                (ilo ilo)
                                ((1 ldh) (1 *))
                                h-%offset%)
    (setf (f2cl-lib:fref wi-%data% (ilo) ((1 *)) wi-%offset%) zero)
    (go end_label)))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(setf nz (f2cl-lib:int-add (f2cl-lib:int-sub ihiz iloz) 1))
(setf unfl (dlamch "Safe minimum"))
(setf ovfl (/ one unfl))
(multiple-value-bind (var-0 var-1)
  (dlabad unfl ovfl)
  (declare (ignore))
  (setf unfl var-0)
  (setf ovfl var-1))
(setf ulp (dlamch "Precision"))
(setf smlnum (* unfl (/ nh ulp)))
(cond
  (wantt
    (setf i1 1)
    (setf i2 n)))
  (setf itn (f2cl-lib:int-mul 30 nh))
  (setf i ihi)
label10
  (setf l ilo)
  (if (< i ilo) (go end_label))
  (f2cl-lib:fdo (its 0 (f2cl-lib:int-add its 1))
    ((> its itn) nil)
    (tagbody
      (f2cl-lib:fdo (k i (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
        ((> k (f2cl-lib:int-add l 1)) nil)
        (tagbody
          (setf tst1
            (+
              (abs
                (f2cl-lib:fref h-%data%
                  ((f2cl-lib:int-sub k 1)
                   (f2cl-lib:int-sub k 1))
                  ((1 ldh) (1 *))
                  h-%offset%))
              (abs
                (f2cl-lib:fref h-%data%
                  (k k)
                  ((1 ldh) (1 *))
                  h-%offset%))))
            (if (= tst1 zero)
              (setf tst1
                (dlanhs "1"
                  (f2cl-lib:int-add (f2cl-lib:int-sub i 1) 1)
                  (f2cl-lib:array-slice h
                    double-float

```

```

                                (1 1)
                                ((1 ldh) (1 *)))
                                ldh work)))
(if
  (<=
    (abs
      (f2cl-lib:fref h-%data%
                     (k (f2cl-lib:int-sub k 1))
                     ((1 ldh) (1 *)))
      h-%offset%))
    (max (* ulp tst1) smlnum))
  (go label30))))
label30
(setf l k)
(cond
  ((> l ilo)
    (setf (f2cl-lib:fref h-%data%
                        (l (f2cl-lib:int-sub l 1))
                        ((1 ldh) (1 *)))
      h-%offset%
      zero)))
  (if (>= 1 (f2cl-lib:int-sub i 1)) (go label140))
  (cond
    ((not wantt)
      (setf i1 l)
      (setf i2 i)))
    (cond
      ((or (= its 10) (= its 20))
        (setf s
          (+
            (abs
              (f2cl-lib:fref h-%data%
                             (i (f2cl-lib:int-sub i 1))
                             ((1 ldh) (1 *)))
              h-%offset%))
            (abs
              (f2cl-lib:fref h-%data%
                             ((f2cl-lib:int-sub i 1)
                              (f2cl-lib:int-sub i 2))
                             ((1 ldh) (1 *)))
              h-%offset%))))))
        (setf h44
          (+ (* dat1 s)
            (f2cl-lib:fref h-%data%
                           (i i)
                           ((1 ldh) (1 *)))
            h-%offset%)))
        (setf h33 h44)
        (setf h43h34 (* dat2 s s)))
      (t

```

```

(setf h44
  (f2cl-lib:fref h-%data%
    (i i)
    ((1 ldh) (1 *))
    h-%offset%))
(setf h33
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-sub i 1)
     (f2cl-lib:int-sub i 1))
    ((1 ldh) (1 *))
    h-%offset%))
(setf h43h34
  (*
    (f2cl-lib:fref h-%data%
      (i (f2cl-lib:int-sub i 1))
      ((1 ldh) (1 *))
      h-%offset%)
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-sub i 1) i)
      ((1 ldh) (1 *))
      h-%offset%)))
(setf s
  (*
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-sub i 1)
       (f2cl-lib:int-sub i 2))
      ((1 ldh) (1 *))
      h-%offset%)
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-sub i 1)
       (f2cl-lib:int-sub i 2))
      ((1 ldh) (1 *))
      h-%offset%)))
(setf disc (* (- h33 h44) half))
(setf disc (+ (* disc disc) h43h34))
(cond
  (> disc zero)
    (setf disc (f2cl-lib:fsqrt disc))
    (setf ave (* half (+ h33 h44)))
    (cond
      (> (+ (abs h33) (- (abs h44))) zero)
        (setf h33 (- (* h33 h44) h43h34))
        (setf h44 (/ h33 (+ (f2cl-lib:sign disc ave) ave))))
      (t
        (setf h44 (+ (f2cl-lib:sign disc ave) ave))))
    (setf h33 h44)
    (setf h43h34 zero))))
(f2cl-lib:fdo (m (f2cl-lib:int-add i (f2cl-lib:int-sub 2))
  (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
  (> m 1) nil)

```

```

(tagbody
  (setf h11
    (f2cl-lib:fref h-%data%
      (m m)
      ((1 ldh) (1 *))
      h-%offset%))

  (setf h22
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add m 1)
       (f2cl-lib:int-add m 1))
      ((1 ldh) (1 *))
      h-%offset%))

  (setf h21
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add m 1) m)
      ((1 ldh) (1 *))
      h-%offset%))

  (setf h12
    (f2cl-lib:fref h-%data%
      (m (f2cl-lib:int-add m 1))
      ((1 ldh) (1 *))
      h-%offset%))

  (setf h44s (- h44 h11))
  (setf h33s (- h33 h11))
  (setf v1 (+ (/ (- (* h33s h44s) h43h34) h21) h12))
  (setf v2 (- h22 h11 h33s h44s))
  (setf v3
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add m 2)
       (f2cl-lib:int-add m 1))
      ((1 ldh) (1 *))
      h-%offset%))

  (setf s (+ (abs v1) (abs v2) (abs v3)))
  (setf v1 (/ v1 s))
  (setf v2 (/ v2 s))
  (setf v3 (/ v3 s))
  (setf (f2cl-lib:fref v (1) ((1 3))) v1)
  (setf (f2cl-lib:fref v (2) ((1 3))) v2)
  (setf (f2cl-lib:fref v (3) ((1 3))) v3)
  (if (= m 1) (go label50))
  (setf h00
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-sub m 1)
       (f2cl-lib:int-sub m 1))
      ((1 ldh) (1 *))
      h-%offset%))

  (setf h10
    (f2cl-lib:fref h-%data%
      (m (f2cl-lib:int-sub m 1))
      ((1 ldh) (1 *))

```

```

                                h-%offset%)
    (setf tst1 (* (abs v1) (+ (abs h00) (abs h11) (abs h22))))
    (if (<= (* (abs h10) (+ (abs v2) (abs v3))) (* ulp tst1))
        (go label50))))
label50
(f2cl-lib:fdo (k m (f2cl-lib:int-add k 1))
    (> k (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
(tagbody
    (setf nr
        (min (the fixnum 3)
            (the fixnum
                (f2cl-lib:int-add (f2cl-lib:int-sub i k)
                    1))))
    (if (> k m)
        (dcopy nr
            (f2cl-lib:array-slice h
                double-float
                (k (f2cl-lib:int-sub k 1))
                ((1 ldh) (1 *)))
            1 v 1))
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
        (dlarfz nr (f2cl-lib:fref v (1) ((1 3)))
            (f2cl-lib:array-slice v double-float (2) ((1 3))) 1 t1)
        (declare (ignore var-0 var-2 var-3))
        (setf (f2cl-lib:fref v (1) ((1 3))) var-1)
        (setf t1 var-4))
    (cond
        ((> k m)
            (setf (f2cl-lib:fref h-%data%
                (k (f2cl-lib:int-sub k 1))
                ((1 ldh) (1 *))
                h-%offset%)
                (f2cl-lib:fref v (1) ((1 3))))
            (setf (f2cl-lib:fref h-%data%
                ((f2cl-lib:int-add k 1)
                 (f2cl-lib:int-sub k 1))
                ((1 ldh) (1 *))
                h-%offset%)
                zero)
            (if (< k (f2cl-lib:int-sub i 1))
                (setf (f2cl-lib:fref h-%data%
                    ((f2cl-lib:int-add k 2)
                     (f2cl-lib:int-sub k 1))
                    ((1 ldh) (1 *))
                    h-%offset%)
                    zero)))
        ((> m 1)
            (setf (f2cl-lib:fref h-%data%
                (k (f2cl-lib:int-sub k 1))
                ((1 ldh) (1 *))
                h-%offset%)
                zero))))

```

```

                                h-%offset%)
(-
  (f2c1-lib:fref h-%data%
    (k (f2c1-lib:int-sub k 1))
    ((1 ldh) (1 *))
    h-%offset%))))))
(setf v2 (f2c1-lib:fref v (2) ((1 3))))
(setf t2 (* t1 v2))
(cond
  ((= nr 3)
   (setf v3 (f2c1-lib:fref v (3) ((1 3))))
   (setf t3 (* t1 v3))
   (f2c1-lib:fdo (j k (f2c1-lib:int-add j 1))
     (> j i2) nil)
   (tagbody
    (setf sum
      (+
        (f2c1-lib:fref h-%data%
          (k j)
          ((1 ldh) (1 *))
          h-%offset%)
        (* v2
          (f2c1-lib:fref h-%data%
            ((f2c1-lib:int-add k 1) j)
            ((1 ldh) (1 *))
            h-%offset%))
        (* v3
          (f2c1-lib:fref h-%data%
            ((f2c1-lib:int-add k 2) j)
            ((1 ldh) (1 *))
            h-%offset%))))))
    (setf (f2c1-lib:fref h-%data%
      (k j)
      ((1 ldh) (1 *))
      h-%offset%)
      (-
        (f2c1-lib:fref h-%data%
          (k j)
          ((1 ldh) (1 *))
          h-%offset%)
        (* sum t1)))
    (setf (f2c1-lib:fref h-%data%
      ((f2c1-lib:int-add k 1) j)
      ((1 ldh) (1 *))
      h-%offset%)
      (-
        (f2c1-lib:fref h-%data%
          ((f2c1-lib:int-add k 1) j)
          ((1 ldh) (1 *))
          h-%offset%)

```

```

(* sum t2)))
(setf (f2cl-lib:fref h-%data%
                    ((f2cl-lib:int-add k 2) j)
                    ((1 ldh) (1 *))
                    h-%offset%)
(-
  (f2cl-lib:fref h-%data%
                ((f2cl-lib:int-add k 2) j)
                ((1 ldh) (1 *))
                h-%offset%)
(* sum t3))))
(f2cl-lib:fdo (j i1 (f2cl-lib:int-add j 1))
  (> j
    (min
      (the fixnum
        (f2cl-lib:int-add k 3))
      (the fixnum i)))
    nil)
(tagbody
  (setf sum
    (+
      (f2cl-lib:fref h-%data%
                    (j k)
                    ((1 ldh) (1 *))
                    h-%offset%)
      (* v2
        (f2cl-lib:fref h-%data%
                      (j (f2cl-lib:int-add k 1))
                      ((1 ldh) (1 *))
                      h-%offset%))
      (* v3
        (f2cl-lib:fref h-%data%
                      (j (f2cl-lib:int-add k 2))
                      ((1 ldh) (1 *))
                      h-%offset%))))
    (setf (f2cl-lib:fref h-%data%
                      (j k)
                      ((1 ldh) (1 *))
                      h-%offset%)
      (-
        (f2cl-lib:fref h-%data%
                      (j k)
                      ((1 ldh) (1 *))
                      h-%offset%)
        (* sum t1)))
    (setf (f2cl-lib:fref h-%data%
                      (j (f2cl-lib:int-add k 1))
                      ((1 ldh) (1 *))
                      h-%offset%)
      (-

```



```

(f2cl-lib:fref h-%data%
  (j (f2cl-lib:int-add k 1))
  ((1 ldh) (1 *))
  h-%offset%)
(* sum t2)))
(setf (f2cl-lib:fref h-%data%
  (j (f2cl-lib:int-add k 2))
  ((1 ldh) (1 *))
  h-%offset%)
(-
  (f2cl-lib:fref h-%data%
    (j (f2cl-lib:int-add k 2))
    ((1 ldh) (1 *))
    h-%offset%)
  (* sum t3))))))
(cond
  (wantz
    (f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
      (> j ihiz) nil)
    (tagbody
      (setf sum
        (+
          (f2cl-lib:fref z-%data%
            (j k)
            ((1 ldz) (1 *))
            z-%offset%)
          (* v2
            (f2cl-lib:fref z-%data%
              (j (f2cl-lib:int-add k 1))
              ((1 ldz) (1 *))
              z-%offset%))
          (* v3
            (f2cl-lib:fref z-%data%
              (j (f2cl-lib:int-add k 2))
              ((1 ldz) (1 *))
              z-%offset%))))))
      (setf (f2cl-lib:fref z-%data%
        (j k)
        ((1 ldz) (1 *))
        z-%offset%)
        (-
          (f2cl-lib:fref z-%data%
            (j k)
            ((1 ldz) (1 *))
            z-%offset%)
          (* sum t1)))
      (setf (f2cl-lib:fref z-%data%
        (j (f2cl-lib:int-add k 1))
        ((1 ldz) (1 *))
        z-%offset%)

```

```

(-
  (f2cl-lib:fref z-%data%
    (j (f2cl-lib:int-add k 1))
    ((1 ldz) (1 *))
    z-%offset%)
  (* sum t2)))
(setf (f2cl-lib:fref z-%data%
  (j (f2cl-lib:int-add k 2))
  ((1 ldz) (1 *))
  z-%offset%)
(-
  (f2cl-lib:fref z-%data%
    (j (f2cl-lib:int-add k 2))
    ((1 ldz) (1 *))
    z-%offset%)
  (* sum t3))))))
(= nr 2)
(f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
  (> j i2) nil)
(tagbody
  (setf sum
    (+
      (f2cl-lib:fref h-%data%
        (k j)
        ((1 ldh) (1 *))
        h-%offset%)
      (* v2
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 1) j)
          ((1 ldh) (1 *))
          h-%offset%))))))
(setf (f2cl-lib:fref h-%data%
  (k j)
  ((1 ldh) (1 *))
  h-%offset%)
(-
  (f2cl-lib:fref h-%data%
    (k j)
    ((1 ldh) (1 *))
    h-%offset%)
  (* sum t1)))
(setf (f2cl-lib:fref h-%data%
  ((f2cl-lib:int-add k 1) j)
  ((1 ldh) (1 *))
  h-%offset%)
(-
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1) j)
    ((1 ldh) (1 *))
    h-%offset%)

```

```

(* sum t2))))))
(f2cl-lib:fdo (j i1 (f2cl-lib:int-add j 1))
  (> j i) nil)

(tagbody
  (setf sum
    (+
      (f2cl-lib:fref h-%data%
        (j k)
        ((1 ldh) (1 *)))
        h-%offset%)

      (* v2
        (f2cl-lib:fref h-%data%
          (j (f2cl-lib:int-add k 1))
          ((1 ldh) (1 *)))
          h-%offset%))))))

(setf (f2cl-lib:fref h-%data%
  (j k)
  ((1 ldh) (1 *)))
  h-%offset%)

(-
  (f2cl-lib:fref h-%data%
    (j k)
    ((1 ldh) (1 *)))
    h-%offset%)

  (* sum t1)))

(setf (f2cl-lib:fref h-%data%
  (j (f2cl-lib:int-add k 1))
  ((1 ldh) (1 *)))
  h-%offset%)

(-
  (f2cl-lib:fref h-%data%
    (j (f2cl-lib:int-add k 1))
    ((1 ldh) (1 *)))
    h-%offset%)

  (* sum t2))))))

(cond
  (wantz
    (f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
      (> j ihiz) nil)

    (tagbody
      (setf sum
        (+
          (f2cl-lib:fref z-%data%
            (j k)
            ((1 ldz) (1 *)))
            z-%offset%)

          (* v2
            (f2cl-lib:fref z-%data%
              (j (f2cl-lib:int-add k 1))
              ((1 ldz) (1 *)))
              z-%offset%))))))

```

```

                                z-%offset%))))
(setf (f2cl-lib:fref z-%data%
                    (j k)
                    ((1 ldz) (1 *))
                    z-%offset%))
(-
  (f2cl-lib:fref z-%data%
                (j k)
                ((1 ldz) (1 *))
                z-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref z-%data%
                    (j (f2cl-lib:int-add k 1))
                    ((1 ldz) (1 *))
                    z-%offset%))
(-
  (f2cl-lib:fref z-%data%
                (j (f2cl-lib:int-add k 1))
                ((1 ldz) (1 *))
                z-%offset%))
  (* sum t2)))))))))

(setf info i)
(go end_label)
label140
(cond
  ((= 1 i)
   (setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
         (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
   (setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) zero))
  ((= 1 (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
     (dlanv2
      (f2cl-lib:fref h-%data%
                    ((f2cl-lib:int-sub i 1) (f2cl-lib:int-sub i 1))
                    ((1 ldh) (1 *))
                    h-%offset%)
      (f2cl-lib:fref h-%data%
                    ((f2cl-lib:int-sub i 1) i)
                    ((1 ldh) (1 *))
                    h-%offset%)
      (f2cl-lib:fref h-%data%
                    (i (f2cl-lib:int-sub i 1))
                    ((1 ldh) (1 *))
                    h-%offset%)
      (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%)
      (f2cl-lib:fref wr-%data%
                    ((f2cl-lib:int-sub i 1))
                    ((1 *))
                    wr-%offset%))

```

```

(f2cl-lib:fref wi-%data%
  ((f2cl-lib:int-sub i 1))
  ((1 *))
  wi-%offset%)
(f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%)
(f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) cs sn)
(declare (ignore))
(setf (f2cl-lib:fref h-%data%
  ((f2cl-lib:int-sub i 1))
  (f2cl-lib:int-sub i 1))
  ((1 ldh) (1 *))
  h-%offset%)

  var-0)
(setf (f2cl-lib:fref h-%data%
  ((f2cl-lib:int-sub i 1) i)
  ((1 ldh) (1 *))
  h-%offset%)

  var-1)
(setf (f2cl-lib:fref h-%data%
  (i (f2cl-lib:int-sub i 1))
  ((1 ldh) (1 *))
  h-%offset%)

  var-2)
(setf (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%)

  var-3)
(setf (f2cl-lib:fref wr-%data%
  ((f2cl-lib:int-sub i 1))
  ((1 *))
  wr-%offset%)

  var-4)
(setf (f2cl-lib:fref wi-%data%
  ((f2cl-lib:int-sub i 1))
  ((1 *))
  wi-%offset%)

  var-5)
(setf (f2cl-lib:fref wr-%data% (i) ((1 *)) wr-%offset%) var-6)
(setf (f2cl-lib:fref wi-%data% (i) ((1 *)) wi-%offset%) var-7)
(setf cs var-8)
(setf sn var-9))
(cond
  (wantt
    (if (> i2 i)
      (drot (f2cl-lib:int-sub i2 i)
        (f2cl-lib:array-slice h
          double-float
          ((+ i (f2cl-lib:int-sub 1))
            (f2cl-lib:int-add i 1))
          ((1 ldh) (1 *)))
        ldh
        (f2cl-lib:array-slice h

```

```

                                double-float
                                (i (f2cl-lib:int-add i 1))
                                ((1 ldh) (1 *)))

                                ldh cs sn))
(drot (f2cl-lib:int-sub i i1 1)
      (f2cl-lib:array-slice h
                            double-float
                            (i1 (f2cl-lib:int-sub i 1))
                            ((1 ldh) (1 *)))
      1 (f2cl-lib:array-slice h double-float (i1 i) ((1 ldh) (1 *))) 1
      cs sn)))
(cond
 (wantz
  (drot nz
        (f2cl-lib:array-slice z
                              double-float
                              (iloz (f2cl-lib:int-sub i 1))
                              ((1 ldz) (1 *)))
        1 (f2cl-lib:array-slice z double-float (iloz i) ((1 ldz) (1 *)))
        1 cs sn))))))
(setf itn (f2cl-lib:int-sub itn its))
(setf i (f2cl-lib:int-sub 1 1))
(go label10)
end_label
(return
 (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dlahrd LAPACK

— dlahrd.input —

```

)set break resume
)sys rm -f dlahrd.output
)spool dlahrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlahrd.help —

```
=====
dlahrd examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLAHRD - the first NB columns of a real general n-by-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero

SYNOPSIS

```
SUBROUTINE DLAHRD( N, K, NB, A, LDA, TAU, T, LDT, Y, LDY )
```

```
      INTEGER      K, LDA, LDT, LDY, N, NB
```

```
      DOUBLE      PRECISION A( LDA, * ), T( LDT, NB ), TAU( NB ), Y(
         LDY, NB )
```

PURPOSE

DLAHRD reduces the first NB columns of a real general n-by-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by an orthogonal similarity transformation $Q' * A * Q$. The routine returns the matrices V and T which determine Q as a block reflector $I - V * T * V'$, and also the matrix $Y = A * V * T$.

This is an OBSOLETE auxiliary routine.
 This routine will be 'deprecated' in a future release.
 Please use the new routine DLAHR2 instead.

ARGUMENTS

N (input) INTEGER
 The order of the matrix A.

K (input) INTEGER
 The offset for the reduction. Elements below the k-th subdiagonal in the first NB columns are reduced to zero.

NB (input) INTEGER
 The number of columns to be reduced.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N-K+1)
 On entry, the n-by-(n-k+1) general matrix A. On exit, the elements on and above the k-th subdiagonal in the first NB columns are overwritten with the corresponding elements of the reduced matrix; the elements below the k-th subdiagonal, with the array TAU, represent the matrix Q as a product of elementary reflectors. The other columns of A are unchanged. See Further

Details. LDA (input) INTEGER The leading dimension of the array A. $LDA \geq \max(1, N)$.

TAU (output) DOUBLE PRECISION array, dimension (NB)
The scalar factors of the elementary reflectors. See Further Details.

T (output) DOUBLE PRECISION array, dimension (LDT, NB)
The upper triangular matrix T.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq NB$.

Y (output) DOUBLE PRECISION array, dimension (LDY, NB)
The n-by-nb matrix Y.

LDY (input) INTEGER
The leading dimension of the array Y. $LDY \geq N$.

FURTHER DETAILS

The matrix Q is represented as a product of nb elementary reflectors

$$Q = H(1) H(2) \dots H(nb).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with $v(1:i+k-1) = 0$, $v(i+k) = 1$; $v(i+k+1:n)$ is stored on exit in $A(i+k+1:n, i)$, and tau in TAU(i).

The elements of the vectors v together form the (n-k+1)-by-nb matrix V which is needed, with T and Y, to apply the transformation to the unreduced part of the matrix, using an update of the form: $A := (I - V * T * V') * (A - Y * V')$.

The contents of A on exit are illustrated by the following example with $n = 7$, $k = 3$ and $nb = 2$:

```
( a  h  a  a  a )
( a  h  a  a  a )
( a  h  a  a  a )
( h  h  a  a  a )
( v1 h  a  a  a )
( v1 v2 a  a  a )
( v1 v2 a  a  a )
```

where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and vi denotes an ele-

ment of the vector defining H(i).

— dlahrd.f —

```

SUBROUTINE DLAHRD( N, K, NB, A, LDA, TAU, T, LDT, Y, LDY )
*
* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   June 30, 1999
*
*   .. Scalar Arguments ..
      INTEGER          K, LDA, LDT, LDY, N, NB
*
*   .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), T( LDT, NB ), TAU( NB ),
$      Y( LDY, NB )
*
*   ..
*
* =====
*
*   .. Parameters ..
      DOUBLE PRECISION  ZERO, ONE
      PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*   ..
*   .. Local Scalars ..
      INTEGER            I
      DOUBLE PRECISION   EI
*
*   ..
*   .. External Subroutines ..
      EXTERNAL           DAXPY, DCOPY, DGEMV, DLARFG, DSCAL, DTRMV
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC           MIN
*
*   ..
*   .. Executable Statements ..
*
*   Quick return if possible
*
      IF( N.LE.1 )
$      RETURN
*
      DO 10 I = 1, NB
          IF( I.GT.1 ) THEN
*
*           Update A(1:n,i)

```

```

*
*      Compute i-th column of  $A - Y * V'$ 
*
*      CALL DGEMV( 'No transpose', N, I-1, -ONE, Y, LDY,
$           A( K+I-1, 1 ), LDA, ONE, A( 1, I ), 1 )
*
*      Apply  $I - V * T' * V'$  to this column (call it b) from the
*      left, using the last column of T as workspace
*
*      Let  $V = \begin{pmatrix} V1 \\ V2 \end{pmatrix}$  and  $b = \begin{pmatrix} b1 \\ b2 \end{pmatrix}$  (first I-1 rows)
*
*      where V1 is unit lower triangular
*
*      w := V1' * b1
*
*      CALL DCOPY( I-1, A( K+1, I ), 1, T( 1, NB ), 1 )
*      CALL DTRMV( 'Lower', 'Transpose', 'Unit', I-1, A( K+1, 1 ),
$           LDA, T( 1, NB ), 1 )
*
*      w := w + V2'*b2
*
*      CALL DGEMV( 'Transpose', N-K-I+1, I-1, ONE, A( K+I, 1 ),
$           LDA, A( K+I, I ), 1, ONE, T( 1, NB ), 1 )
*
*      w := T'*w
*
*      CALL DTRMV( 'Upper', 'Transpose', 'Non-unit', I-1, T, LDT,
$           T( 1, NB ), 1 )
*
*      b2 := b2 - V2*w
*
*      CALL DGEMV( 'No transpose', N-K-I+1, I-1, -ONE, A( K+I, 1 ),
$           LDA, T( 1, NB ), 1, ONE, A( K+I, I ), 1 )
*
*      b1 := b1 - V1*w
*
*      CALL DTRMV( 'Lower', 'No transpose', 'Unit', I-1,
$           A( K+1, 1 ), LDA, T( 1, NB ), 1 )
*      CALL DAXPY( I-1, -ONE, T( 1, NB ), 1, A( K+1, I ), 1 )
*
*      A( K+I-1, I-1 ) = EI
*      END IF
*
*      Generate the elementary reflector H(i) to annihilate
*      A(k+i+1:n,i)
*
*      CALL DLARFG( N-K-I+1, A( K+I, I ), A( MIN( K+I+1, N ), I ), 1,
$           TAU( I ) )
*      EI = A( K+I, I )

```

```

      A( K+I, I ) = ONE
*
*      Compute Y(1:n,i)
*
      CALL DGEMV( 'No transpose', N, N-K-I+1, ONE, A( 1, I+1 ), LDA,
$           A( K+I, I ), 1, ZERO, Y( 1, I ), 1 )
      CALL DGEMV( 'Transpose', N-K-I+1, I-1, ONE, A( K+I, 1 ), LDA,
$           A( K+I, I ), 1, ZERO, T( 1, I ), 1 )
      CALL DGEMV( 'No transpose', N, I-1, -ONE, Y, LDY, T( 1, I ), 1,
$           ONE, Y( 1, I ), 1 )
      CALL DSCAL( N, TAU( I ), Y( 1, I ), 1 )
*
*      Compute T(1:i,i)
*
      CALL DSCAL( I-1, -TAU( I ), T( 1, I ), 1 )
      CALL DTRMV( 'Upper', 'No transpose', 'Non-unit', I-1, T, LDT,
$           T( 1, I ), 1 )
      T( I, I ) = TAU( I )
*
10 CONTINUE
      A( K+NB, NB ) = EI
*
      RETURN
*
*      End of DLAHRD
*
      END

```

— LAPACK dlahrd —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlahrd (n k nb a lda tau t$ ldt y ldy)
    (declare (type (simple-array double-float (*)) y t$ tau a)
              (type fixnum ldy ldt lda nb k n))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (t$ double-float t$-%data% t$-%offset%)
       (y double-float y-%data% y-%offset%))
      (prog ((ei 0.0) (i 0))
        (declare (type (double-float) ei) (type fixnum i))
        (if (<= n 1) (go end_label))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i nb) nil)

```

```

(tagbody
(cond
  (> i 1)
  (dgemv "No transpose" n (f2cl-lib:int-sub i 1) (- one) y ldy
    (f2cl-lib:array-slice a
      double-float
      ((+ k i (f2cl-lib:int-sub 1)) 1)
      ((1 lda) (1 *)))

  lda one
  (f2cl-lib:array-slice a double-float (1 i) ((1 lda) (1 *))) 1)
(dcopy (f2cl-lib:int-sub i 1)
  (f2cl-lib:array-slice a
    double-float
    ((+ k 1) i)
    ((1 lda) (1 *)))

  1
  (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
  1)
(dtrmv "Lower" "Transpose" "Unit" (f2cl-lib:int-sub i 1)
  (f2cl-lib:array-slice a
    double-float
    ((+ k 1) 1)
    ((1 lda) (1 *)))

  lda
  (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
  1)
(dgemv "Transpose" (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
  (f2cl-lib:int-sub i 1) one
  (f2cl-lib:array-slice a
    double-float
    ((+ k i) 1)
    ((1 lda) (1 *)))

  lda
  (f2cl-lib:array-slice a
    double-float
    ((+ k i) i)
    ((1 lda) (1 *)))

  1 one
  (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
  1)
(dtrmv "Upper" "Transpose" "Non-unit" (f2cl-lib:int-sub i 1) t$
  ldt
  (f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
  1)
(dgemv "No transpose"
  (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
  (f2cl-lib:int-sub i 1) (- one)
  (f2cl-lib:array-slice a
    double-float
    ((+ k i) 1)

```

```

                                ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
1 one
(f2cl-lib:array-slice a
  double-float
  ((+ k i) i)
  ((1 lda) (1 *)))
1)
(dtrmv "Lower" "No transpose" "Unit" (f2cl-lib:int-sub i 1)
(f2cl-lib:array-slice a
  double-float
  ((+ k 1) 1)
  ((1 lda) (1 *)))
lda
(f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
1)
(daxpy (f2cl-lib:int-sub i 1) (- one)
(f2cl-lib:array-slice t$ double-float (1 nb) ((1 ldt) (1 nb)))
1
(f2cl-lib:array-slice a
  double-float
  ((+ k 1) i)
  ((1 lda) (1 *)))
1)
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-sub (f2cl-lib:int-add k i)
    1)
  (f2cl-lib:int-sub i 1))
  ((1 lda) (1 *))
  a-%offset%)
  ei)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlarf (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
  (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-add k i) i)
    ((1 lda) (1 *))
    a-%offset%)
  (f2cl-lib:array-slice a
    double-float
    ((min (f2cl-lib:int-add k i 1) n) i)
    ((1 lda) (1 *)))
  1 (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
(declare (ignore var-0 var-2 var-3))
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-add k i) i)
  ((1 lda) (1 *))
  a-%offset%)
  var-1)
(setf (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%)

```

```

                                var-4))
(setf ei
  (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-add k i) i)
    ((1 lda) (1 *))
    a-%offset%))
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-add k i) i)
  ((1 lda) (1 *))
  a-%offset%))

  one)
(dgemv "No transpose" n
  (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1) one
  (f2cl-lib:array-slice a
    double-float
    (1 (f2cl-lib:int-add i 1))
    ((1 lda) (1 *))))

lda
(f2cl-lib:array-slice a double-float ((+ k i) i) ((1 lda) (1 *)))
1 zero
(f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb))) 1)
(dgemv "Transpose" (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
  (f2cl-lib:int-sub i 1) one
  (f2cl-lib:array-slice a double-float ((+ k i) 1) ((1 lda) (1 *)))
  lda
  (f2cl-lib:array-slice a double-float ((+ k i) i) ((1 lda) (1 *)))
  1 zero
  (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1)
(dgemv "No transpose" n (f2cl-lib:int-sub i 1) (- one) y ldy
  (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1
  one (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb)))
  1)
(dscal n (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%)
  (f2cl-lib:array-slice y double-float (1 i) ((1 ldy) (1 nb))) 1)
(dscal (f2cl-lib:int-sub i 1)
  (- (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
  (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb))) 1)
(dtrmv "Upper" "No transpose" "Non-unit" (f2cl-lib:int-sub i 1) t$
  ldt (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 nb)))
  1)
(setf (f2cl-lib:fref t$-%data% (i i) ((1 ldt) (1 nb)) t$-%offset%)
  (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%)))
(setf (f2cl-lib:fref a-%data%
  ((f2cl-lib:int-add k nb) nb)
  ((1 lda) (1 *))
  a-%offset%))

  ei)
end_label
(return (values nil nil nil nil nil nil nil nil nil nil))))))

```

dlaisnan LAPACK

— dlaisnan.input —

```
)set break resume
)sys rm -f dlaisnan.output
)spool dlaisnan.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlaisnan.help —

```
=====
dlaisnan examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
LOGICAL FUNCTION DLAISNAN( DIN1, DIN2 )

.. Scalar Arguments ..
DOUBLE PRECISION   DIN1, DIN2
..
```

Purpose:
 =====

This routine is not for general use. It exists solely to avoid over-optimization in DISNAN.

DLAISNAN checks for NaNs by comparing its two arguments for inequality. NaN is the only floating-point value where NaN != NaN returns .TRUE. To check for NaNs, pass the same variable as both arguments.

A compiler must assume that the two arguments are not the same variable, and the test will not be optimized away. Interprocedural or whole-program optimization may delete this test. The ISNAN functions will be replaced by the correct Fortran 03 intrinsic once the intrinsic is widely available.

Arguments:
=====

[in] DIN1
DIN1 is DOUBLE PRECISION

[in] DIN2
DIN2 is DOUBLE PRECISION
Two numbers to compare for inequality.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— dlaisnan.f —

```
* =====
* LOGICAL FUNCTION DLAISNAN( DIN1, DIN2 )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
* November 2011
*
* .. Scalar Arguments ..
* DOUBLE PRECISION DIN1, DIN2
* ..
*
* =====
```



```

*
* .. Executable Statements ..
      DLAISNAN = (DIN1.NE.DIN2)
      RETURN
      END

```

— LAPACK dlaisnan —

dlaln2 LAPACK

— dlaln2.input —

```

)set break resume
)sys rm -f dlaln2.output
)spool dlaln2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlaln2.help —

```

=====
dlaln2 examples
=====

=====
Man Page Details
=====

```

NAME

DLALN2 - a system of the form $(\alpha A - w D) X = s B$ or $(\alpha A' - w D) X = s B$ with possible scaling ("s") and perturbation of A

SYNOPSIS

```

SUBROUTINE DLALN2( LTRANS, NA, NW, SMIN, CA, A, LDA, D1, D2, B, LDB,
                  WR, WI, X, LDX, SCALE, XNORM, INFO )

```

| | |
|---------|--|
| LOGICAL | LTRANS |
| INTEGER | INFO, LDA, LDB, LDX, NA, NW |
| DOUBLE | PRECISION CA, D1, D2, SCALE, SMIN, WI, WR, XNORM |
| DOUBLE | PRECISION A(LDA, *), B(LDB, *), X(LDX, *) |

Purpose

```

=====

```

DLALN2 solves a system of the form $(ca A - w D) X = s B$ or $(ca A' - w D) X = s B$ with possible scaling ("s") and perturbation of A. (A' means A-transpose.)

A is an NA x NA real matrix, ca is a real scalar, D is an NA x NA real diagonal matrix, w is a real or complex value, and X and B are NA x 1 matrices -- real if w is real, complex if w is complex. NA may be 1 or 2.

If w is complex, X and B are represented as NA x 2 matrices, the first column of each being the real part and the second being the imaginary part.

"s" is a scaling factor (LE. 1), computed by DLALN2, which is so chosen that X can be computed without overflow. X is further scaled if necessary to assure that $\text{norm}(ca A - w D) * \text{norm}(X)$ is less than overflow.

If both singular values of $(ca A - w D)$ are less than SMIN, SMIN*identity will be used instead of $(ca A - w D)$. If only one singular value is less than SMIN, one element of $(ca A - w D)$ will be perturbed enough to make the smallest singular value roughly SMIN. If both singular values are at least SMIN, $(ca A - w D)$ will not be perturbed. In any case, the perturbation will be at most some small multiple of $\max(\text{SMIN}, \text{ulp} * \text{norm}(ca A - w D))$. The singular values are computed by infinity-norm approximations, and thus will only be correct to a factor of 2 or so.

Note: all input quantities are assumed to be smaller than overflow by a reasonable factor. (See BIGNUM.)

Arguments

```

=====

```

LTRANS (input) LOGICAL

=.TRUE.: A-transpose will be used.
=.FALSE.: A will be used (not transposed.)

NA (input) INTEGER
The size of the matrix A. It may (only) be 1 or 2.

NW (input) INTEGER
1 if "w" is real, 2 if "w" is complex. It may only be 1 or 2.

SMIN (input) DOUBLE PRECISION
The desired lower bound on the singular values of A. This should be a safe distance away from underflow or overflow, say, between (underflow/machine precision) and (machine precision * overflow). (See BIGNUM and ULP.)

CA (input) DOUBLE PRECISION
The coefficient c, which A is multiplied by.

A (input) DOUBLE PRECISION array, dimension (LDA,NA)
The NA x NA matrix A.

LDA (input) INTEGER
The leading dimension of A. It must be at least NA.

D1 (input) DOUBLE PRECISION
The 1,1 element in the diagonal matrix D.

D2 (input) DOUBLE PRECISION
The 2,2 element in the diagonal matrix D. Not used if NW=1.

B (input) DOUBLE PRECISION array, dimension (LDB,NW)
The NA x NW matrix B (right-hand side). If NW=2 ("w" is complex), column 1 contains the real part of B and column 2 contains the imaginary part.

LDB (input) INTEGER
The leading dimension of B. It must be at least NA.

WR (input) DOUBLE PRECISION
The real part of the scalar "w".

WI (input) DOUBLE PRECISION
The imaginary part of the scalar "w". Not used if NW=1.

X (output) DOUBLE PRECISION array, dimension (LDX,NW)
The NA x NW matrix X (unknowns), as computed by DLALN2. If NW=2 ("w" is complex), on exit, column 1 will contain the real part of X and column 2 will contain the imaginary part.

LDX (input) INTEGER
 The leading dimension of X. It must be at least NA.

SCALE (output) DOUBLE PRECISION
 The scale factor that B must be multiplied by to insure
 that overflow does not occur when computing X. Thus,
 (ca A - w D) X will be SCALE*B, not B (ignoring
 perturbations of A.) It will be at most 1.

XNORM (output) DOUBLE PRECISION
 The infinity-norm of X, when X is regarded as an NA x NW
 real matrix.

INFO (output) INTEGER
 An error flag. It will be set to zero if no error occurs,
 a negative number if an argument is in error, or a positive
 number if ca A - w D had to be perturbed.
 The possible values are:
 = 0: No error occurred, and (ca A - w D) did not have to be
 perturbed.
 = 1: (ca A - w D) had to be perturbed to make its smallest
 (or only) singular value greater than SMIN.
 NOTE: In the interests of speed, this routine does not
 check the inputs for errors.

— dlaln2.f —

```

      SUBROUTINE DLALN2( LTRANS, NA, NW, SMIN, CA, A, LDA, D1, D2, B,
$                      LDB, WR, WI, X, LDX, SCALE, XNORM, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1992
*
*    .. Scalar Arguments ..
      LOGICAL          LTRANS
      INTEGER          INFO, LDA, LDB, LDX, NA, NW
      DOUBLE PRECISION CA, D1, D2, SCALE, SMIN, WI, WR, XNORM
*
*    ..
*
*    .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), B( LDB, * ), X( LDX, * )
*
*    ..
*
*  =====

```

```

*
* .. Parameters ..
DOUBLE PRECISION    ZERO, ONE
PARAMETER            ( ZERO = 0.0D0, ONE = 1.0D0 )
DOUBLE PRECISION    TWO
PARAMETER            ( TWO = 2.0D0 )
*
* ..
* .. Local Scalars ..
INTEGER              ICMAX, J
DOUBLE PRECISION     BBND, BI1, BI2, BIGNUM, BNORM, BR1, BR2, CI21,
$                   CI22, CMAX, CNORM, CR21, CR22, CSI, CSR, LI21,
$                   LR21, SMINI, SMLNUM, TEMP, U22ABS, UI11, UI11R,
$                   UI12, UI12S, UI22, UR11, UR11R, UR12, UR12S,
$                   UR22, XI1, XI2, XR1, XR2
*
* ..
* .. Local Arrays ..
LOGICAL              RSWAP( 4 ), ZSWAP( 4 )
INTEGER              IPIVOT( 4, 4 )
DOUBLE PRECISION     CI( 2, 2 ), CIV( 4 ), CR( 2, 2 ), CRV( 4 )
*
* ..
* .. External Functions ..
DOUBLE PRECISION     DLAMCH
EXTERNAL             DLAMCH
*
* ..
* .. External Subroutines ..
EXTERNAL             DLADIV
*
* ..
* .. Intrinsic Functions ..
INTRINSIC            ABS, MAX
*
* ..
* .. Equivalences ..
C
C *** F2CL cannot currently handle equivalences of arrays
C *** So we do this by hand. Since Fortran arrays are column-major
C *** order, we have the following:
C ***
C *** ci(1,1) civ(1)
C *** ci(2,1) civ(2)
C *** ci(1,2) civ(3)
C *** ci(2,2) civ(4)
C ***
C *** Similarly for CR.
C   EQUIVALENCE      ( CI( 1, 1 ), CIV( 1 ) ),
C   $                ( CR( 1, 1 ), CRV( 1 ) )
*
* ..
* .. Data statements ..
DATA                ZSWAP / .FALSE., .FALSE., .TRUE., .TRUE. /
DATA                RSWAP / .FALSE., .TRUE., .FALSE., .TRUE. /
DATA                IPIVOT / 1, 2, 3, 4, 2, 1, 4, 3, 3, 4, 1, 2, 4,
$                   3, 2, 1 /

```

```

*      ..
*      .. Executable Statements ..
*
*      Compute BIGNUM
*
      SMLNUM = TWO*DLAMCH( 'Safe minimum' )
      BIGNUM = ONE / SMLNUM
      SMINI = MAX( SMIN, SMLNUM )
*
*      Don't check for input errors
*
      INFO = 0
*
*      Standard Initializations
*
      SCALE = ONE
*
      IF( NA.EQ.1 ) THEN
*
*          1 x 1 (i.e., scalar) system    C X = B
*
          IF( NW.EQ.1 ) THEN
*
*              Real 1x1 system.
*
*              C = ca A - w D
*
*              CSR = CA*A( 1, 1 ) - WR*D1
*              CNORM = ABS( CSR )
*
*              If | C | < SMINI, use C = SMINI
*
*              IF( CNORM.LT.SMINI ) THEN
*                  CSR = SMINI
*                  CNORM = SMINI
*                  INFO = 1
*              END IF
*
*              Check scaling for X = B / C
*
*              BNORM = ABS( B( 1, 1 ) )
*              IF( CNORM.LT.ONE .AND. BNORM.GT.ONE ) THEN
*                  IF( BNORM.GT.BIGNUM*CNORM )
$                      SCALE = ONE / BNORM
*              END IF
*
*              Compute X
*
*              X( 1, 1 ) = ( B( 1, 1 )*SCALE ) / CSR
*              XNORM = ABS( X( 1, 1 ) )

```

```

      ELSE
*
*       Complex 1x1 system (w is complex)
*
*       C = ca A - w D
*
      CSR = CA*A( 1, 1 ) - WR*D1
      CSI = -WI*D1
      CNORM = ABS( CSR ) + ABS( CSI )
*
*       If | C | < SMINI, use C = SMINI
*
      IF( CNORM.LT.SMINI ) THEN
        CSR = SMINI
        CSI = ZERO
        CNORM = SMINI
        INFO = 1
      END IF
*
*       Check scaling for X = B / C
*
      BNORM = ABS( B( 1, 1 ) ) + ABS( B( 1, 2 ) )
      IF( CNORM.LT.ONE .AND. BNORM.GT.ONE ) THEN
        IF( BNORM.GT.BIGNUM*CNORM )
$          SCALE = ONE / BNORM
      END IF
*
*       Compute X
*
      CALL DLADIV( SCALE*B( 1, 1 ), SCALE*B( 1, 2 ), CSR, CSI,
$              X( 1, 1 ), X( 1, 2 ) )
      XNORM = ABS( X( 1, 1 ) ) + ABS( X( 1, 2 ) )
      END IF
*
      ELSE
*
*       2x2 System
*
*       Compute the real part of C = ca A - w D (or ca A' - w D )
*
      *** F2CL original
      CR( 1, 1 ) = CA*A( 1, 1 ) - WR*D1
      CR( 2, 2 ) = CA*A( 2, 2 ) - WR*D2
      *** F2CL replacement
      crv(1) = CA*A( 1, 1 ) - WR*D1
      crv(4) = CA*A( 2, 2 ) - WR*D2
      IF( LTRANS ) THEN
c        CR( 1, 2 ) = CA*A( 2, 1 )
c        CR( 2, 1 ) = CA*A( 1, 2 )
        crv( 3 ) = CA*A( 2, 1 )

```

```

        crv( 2 ) = CA*A( 1, 2 )
    ELSE
c        CR( 2, 1 ) = CA*A( 2, 1 )
c        CR( 1, 2 ) = CA*A( 1, 2 )
        crv( 2 ) = CA*A( 2, 1 )
        crv( 3 ) = CA*A( 1, 2 )
    END IF

*
    IF( NW.EQ.1 ) THEN

*
*       Real 2x2 system  (w is real)
*
*       Find the largest element in C
*
        CMAX = ZERO
        ICMAX = 0

*
        DO 10 J = 1, 4
            IF( ABS( CRV( J ) ).GT.CMAX ) THEN
                CMAX = ABS( CRV( J ) )
                ICMAX = J
            END IF
10        CONTINUE

*
*       If norm(C) < SMINI, use SMINI*identity.
*
        IF( CMAX.LT.SMINI ) THEN
            BNORM = MAX( ABS( B( 1, 1 ) ), ABS( B( 2, 1 ) ) )
            IF( SMINI.LT.ONE .AND. BNORM.GT.ONE ) THEN
                IF( BNORM.GT.BIGNUM*SMINI )
$                SCALE = ONE / BNORM
            END IF
            TEMP = SCALE / SMINI
            X( 1, 1 ) = TEMP*B( 1, 1 )
            X( 2, 1 ) = TEMP*B( 2, 1 )
            XNORM = TEMP*BNORM
            INFO = 1
            RETURN
        END IF

*
*       Gaussian elimination with complete pivoting.
*
        UR11 = CRV( ICMAX )
        CR21 = CRV( IPIVOT( 2, ICMAX ) )
        UR12 = CRV( IPIVOT( 3, ICMAX ) )
        CR22 = CRV( IPIVOT( 4, ICMAX ) )
        UR11R = ONE / UR11
        LR21 = UR11R*CR21
        UR22 = CR22 - UR12*LR21

*

```



```

*          If smaller pivot < SMINI, use SMINI
*
      IF( ABS( UR22 ).LT.SMINI ) THEN
        UR22 = SMINI
        INFO = 1
      END IF
      IF( RSWAP( ICMAX ) ) THEN
        BR1 = B( 2, 1 )
        BR2 = B( 1, 1 )
      ELSE
        BR1 = B( 1, 1 )
        BR2 = B( 2, 1 )
      END IF
      BR2 = BR2 - LR21*BR1
      BBND = MAX( ABS( BR1*( UR22*UR11R ) ), ABS( BR2 ) )
      IF( BBND.GT.ONE .AND. ABS( UR22 ).LT.ONE ) THEN
        IF( BBND.GE.BIGNUM*ABS( UR22 ) )
$          SCALE = ONE / BBND
      END IF
*
      XR2 = ( BR2*SCALE ) / UR22
      XR1 = ( SCALE*BR1 )*UR11R - XR2*( UR11R*UR12 )
      IF( ZSWAP( ICMAX ) ) THEN
        X( 1, 1 ) = XR2
        X( 2, 1 ) = XR1
      ELSE
        X( 1, 1 ) = XR1
        X( 2, 1 ) = XR2
      END IF
      XNORM = MAX( ABS( XR1 ), ABS( XR2 ) )
*
*          Further scaling if norm(A) norm(X) > overflow
*
      IF( XNORM.GT.ONE .AND. CMAX.GT.ONE ) THEN
        IF( XNORM.GT.BIGNUM / CMAX ) THEN
          TEMP = CMAX / BIGNUM
          X( 1, 1 ) = TEMP*X( 1, 1 )
          X( 2, 1 ) = TEMP*X( 2, 1 )
          XNORM = TEMP*XNORM
          SCALE = TEMP*SCALE
        END IF
      END IF
    ELSE
*
*          Complex 2x2 system (w is complex)
*
*          Find the largest element in C
*
c          *** F2CL original
c          CI( 1, 1 ) = -WI*D1

```

```

c      CI( 2, 1 ) = ZERO
c      CI( 1, 2 ) = ZERO
c      CI( 2, 2 ) = -WI*D2
      civ( 1 ) = -WI*D1
      civ( 2 ) = ZERO
      civ( 3 ) = ZERO
      civ( 4 ) = -WI*D2
      CMAX = ZERO
      ICMAX = 0

*
      DO 20 J = 1, 4
        IF( ABS( CRV( J ) )+ABS( CIV( J ) ).GT.CMAX ) THEN
          CMAX = ABS( CRV( J ) ) + ABS( CIV( J ) )
          ICMAX = J
        END IF
20    CONTINUE
*
*      If norm(C) < SMINI, use SMINI*identity.
*
      IF( CMAX.LT.SMINI ) THEN
        BNORM = MAX( ABS( B( 1, 1 ) )+ABS( B( 1, 2 ) ),
$          ABS( B( 2, 1 ) )+ABS( B( 2, 2 ) ) )
        IF( SMINI.LT.ONE .AND. BNORM.GT.ONE ) THEN
$          IF( BNORM.GT.BIGNUM*SMINI )
            SCALE = ONE / BNORM
        END IF
        TEMP = SCALE / SMINI
        X( 1, 1 ) = TEMP*B( 1, 1 )
        X( 2, 1 ) = TEMP*B( 2, 1 )
        X( 1, 2 ) = TEMP*B( 1, 2 )
        X( 2, 2 ) = TEMP*B( 2, 2 )
        XNORM = TEMP*BNORM
        INFO = 1
        RETURN
      END IF

*
*      Gaussian elimination with complete pivoting.
*
      UR11 = CRV( ICMAX )
      UI11 = CIV( ICMAX )
      CR21 = CRV( IPIVOT( 2, ICMAX ) )
      CI21 = CIV( IPIVOT( 2, ICMAX ) )
      UR12 = CRV( IPIVOT( 3, ICMAX ) )
      UI12 = CIV( IPIVOT( 3, ICMAX ) )
      CR22 = CRV( IPIVOT( 4, ICMAX ) )
      CI22 = CIV( IPIVOT( 4, ICMAX ) )
      IF( ICMAX.EQ.1 .OR. ICMAX.EQ.4 ) THEN

*
*      Code when off-diagonals of pivoted C are real
*

```

```

      IF( ABS( UR11 ).GT.ABS( UI11 ) ) THEN
        TEMP = UI11 / UR11
        UR11R = ONE / ( UR11*( ONE+TEMP**2 ) )
        UI11R = -TEMP*UR11R
      ELSE
        TEMP = UR11 / UI11
        UI11R = -ONE / ( UI11*( ONE+TEMP**2 ) )
        UR11R = -TEMP*UI11R
      END IF
      LR21 = CR21*UR11R
      LI21 = CR21*UI11R
      UR12S = UR12*UR11R
      UI12S = UR12*UI11R
      UR22 = CR22 - UR12*LR21
      UI22 = CI22 - UR12*LI21
    ELSE
*
*       Code when diagonals of pivoted C are real
*
      UR11R = ONE / UR11
      UI11R = ZERO
      LR21 = CR21*UR11R
      LI21 = CI21*UR11R
      UR12S = UR12*UR11R
      UI12S = UI12*UR11R
      UR22 = CR22 - UR12*LR21 + UI12*LI21
      UI22 = -UR12*LI21 - UI12*LR21
    END IF
    U22ABS = ABS( UR22 ) + ABS( UI22 )
*
*       If smaller pivot < SMINI, use SMINI
*
    IF( U22ABS.LT.SMINI ) THEN
      UR22 = SMINI
      UI22 = ZERO
      INFO = 1
    END IF
    IF( RSWAP( ICMAX ) ) THEN
      BR2 = B( 1, 1 )
      BR1 = B( 2, 1 )
      BI2 = B( 1, 2 )
      BI1 = B( 2, 2 )
    ELSE
      BR1 = B( 1, 1 )
      BR2 = B( 2, 1 )
      BI1 = B( 1, 2 )
      BI2 = B( 2, 2 )
    END IF
    BR2 = BR2 - LR21*BR1 + LI21*BI1
    BI2 = BI2 - LI21*BR1 - LR21*BI1

```

```

      BBND = MAX( ( ABS( BR1 )+ABS( BI1 ) ) *
$          ( U22ABS*( ABS( UR11R )+ABS( UI11R ) ) ),
$          ABS( BR2 )+ABS( BI2 ) )
      IF( BBND.GT.ONE .AND. U22ABS.LT.ONE ) THEN
          IF( BBND.GE.BIGNUM*U22ABS ) THEN
              SCALE = ONE / BBND
              BR1 = SCALE*BR1
              BI1 = SCALE*BI1
              BR2 = SCALE*BR2
              BI2 = SCALE*BI2
          END IF
      END IF
*
      CALL DLADIV( BR2, BI2, UR22, UI22, XR2, XI2 )
      XR1 = UR11R*BR1 - UI11R*BI1 - UR12S*XR2 + UI12S*XI2
      XI1 = UI11R*BR1 + UR11R*BI1 - UI12S*XR2 - UR12S*XI2
      IF( ZSWAP( ICMAX ) ) THEN
          X( 1, 1 ) = XR2
          X( 2, 1 ) = XR1
          X( 1, 2 ) = XI2
          X( 2, 2 ) = XI1
      ELSE
          X( 1, 1 ) = XR1
          X( 2, 1 ) = XR2
          X( 1, 2 ) = XI1
          X( 2, 2 ) = XI2
      END IF
      XNORM = MAX( ABS( XR1 )+ABS( XI1 ), ABS( XR2 )+ABS( XI2 ) )
*
*      Further scaling if  norm(A) norm(X) > overflow
*
      IF( XNORM.GT.ONE .AND. CMAX.GT.ONE ) THEN
          IF( XNORM.GT.BIGNUM / CMAX ) THEN
              TEMP = CMAX / BIGNUM
              X( 1, 1 ) = TEMP*X( 1, 1 )
              X( 2, 1 ) = TEMP*X( 2, 1 )
              X( 1, 2 ) = TEMP*X( 1, 2 )
              X( 2, 2 ) = TEMP*X( 2, 2 )
              XNORM = TEMP*XNORM
              SCALE = TEMP*SCALE
          END IF
      END IF
      END IF
      END IF
*
      RETURN
*
*      End of DLALN2
*
      END

```

```

(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two))
  (let ((zswap
        (make-array 4 :element-type 't :initial-contents '(nil nil t t))
        (rswap
         (make-array 4 :element-type 't :initial-contents '(nil t nil t))
         (ipivot
          (make-array 16
                       :element-type 'fixnum
                       :initial-contents '(1 2 3 4 2 1 4 3 3 4 1 2 4 3 2 1))))))
    (declare (type (simple-array fixnum (16)) ipivot)
              (type (simple-array (member t nil) (4)) rswap zswap))
    (defun dlaln2
      (ltrans na nw smin ca a lda d1 d2 b ldb$ wr wi x ldx scale xnorm
       info)
      (declare (type (simple-array double-float (*)) x b a)
                (type (double-float) xnorm scale wi wr d2 d1 ca smin)
                (type fixnum info ldx ldb$ lda nw na)
                (type (member t nil) ltrans))
      (f2cl-lib:with-multi-array-data
        ((a double-float a-%data% a-%offset%)
         (b double-float b-%data% b-%offset%)
         (x double-float x-%data% x-%offset%))
        (prog ((ci (make-array 4 :element-type 'double-float))
               (civ (make-array 4 :element-type 'double-float))
               (cr (make-array 4 :element-type 'double-float))
               (crv (make-array 4 :element-type 'double-float)) (bbnd 0.0)
               (bi1 0.0) (bi2 0.0) (bignum 0.0) (bnorm 0.0) (br1 0.0) (br2 0.0)
               (ci21 0.0) (ci22 0.0) (cmax 0.0) (cnorm 0.0) (cr21 0.0)
               (cr22 0.0) (csi 0.0) (csr 0.0) (li21 0.0) (lr21 0.0) (smini 0.0)
               (smlnum 0.0) (temp 0.0) (u22abs 0.0) (ui11 0.0) (ui11r 0.0)
               (ui12 0.0) (ui12s 0.0) (ui22 0.0) (ur11 0.0) (ur11r 0.0)
               (ur12 0.0) (ur12s 0.0) (ur22 0.0) (xi1 0.0) (xi2 0.0) (xr1 0.0)
               (xr2 0.0) (icmax 0) (j 0))
              (declare (type (simple-array double-float (4)) ci civ cr crv)
                        (type (double-float) bbnd bi1 bi2 bignum bnorm br1 br2 ci21
                               ci22 cmax cnorm cr21 cr22 csi csr li21
                               lr21 smini smlnum temp u22abs ui11
                               ui11r ui12 ui12s ui22 ur11 ur11r ur12
                               ur12s ur22 xi1 xi2 xr1 xr2)
                        (type fixnum icmax j))

```

```

(setf smlnum (* two (dlamch "Safe minimum")))
(setf bignum (/ one smlnum))
(setf smini (max smin smlnum))
(setf info 0)
(setf scale one)
(cond
  ((= na 1)
    (cond
      ((= nw 1)
        (setf csr
          (-
            (* ca
              (f2cl-lib:fref a-%data%
                (1 1)
                ((1 lda) (1 *))
                a-%offset%))
            (* wr d1)))
        (setf cnorm (abs csr))
        (cond
          ((< cnorm smini)
            (setf csr smini)
            (setf cnorm smini)
            (setf info 1)))
        (setf bnorm
          (abs
            (f2cl-lib:fref b-%data%
              (1 1)
              ((1 ldb$) (1 *))
              b-%offset%)))
        (cond
          ((and (< cnorm one) (> bnorm one)
            (if (> bnorm (* bignum cnorm)) (setf scale (/ one bnorm))))
          (setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
            (/
              (*
                (f2cl-lib:fref b-%data%
                  (1 1)
                  ((1 ldb$) (1 *))
                  b-%offset%))
                scale)
              csr))
        (setf xnorm
          (abs
            (f2cl-lib:fref x-%data%
              (1 1)
              ((1 ldx) (1 *))
              x-%offset%)))
        (t
          (setf csr
            (-

```

```

      (* ca
        (f2cl-lib:fref a-%data%
                      (1 1)
                      ((1 lda) (1 *))
                      a-%offset%))

      (* wr d1)))
(setf csi (* (- wi) d1))
(setf cnorm (+ (abs csr) (abs csi)))
(cond
  ((< cnorm smini)
   (setf csr smini)
   (setf csi zero)
   (setf cnorm smini)
   (setf info 1)))
(setf bnorm
  (+
   (abs
    (f2cl-lib:fref b-%data%
                  (1 1)
                  ((1 ldb$) (1 *))
                  b-%offset%))

   (abs
    (f2cl-lib:fref b-%data%
                  (1 2)
                  ((1 ldb$) (1 *))
                  b-%offset%))))
(cond
  ((and (< cnorm one) (> bnorm one))
   (if (> bnorm (* bignum cnorm)) (setf scale (/ one bnorm))))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv
     (* scale
      (f2cl-lib:fref b-%data%
                    (1 1)
                    ((1 ldb$) (1 *))
                    b-%offset%))

     (* scale
      (f2cl-lib:fref b-%data%
                    (1 2)
                    ((1 ldb$) (1 *))
                    b-%offset%))

     csr csi
     (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
     (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%))
    (declare (ignore var-0 var-1 var-2 var-3))
    (setf (f2cl-lib:fref x-%data%
                      (1 1)
                      ((1 ldx) (1 *))
                      x-%offset%)
          var-4)
    var-4)

```

```

      (setf (f2cl-lib:fref x-%data%
                          (1 2)
                          ((1 lda) (1 *)))
            x-%offset%)
      var-5))
(setf xnorm
      (+
        (abs
          (f2cl-lib:fref x-%data%
                        (1 1)
                        ((1 lda) (1 *)))
          x-%offset%))
        (abs
          (f2cl-lib:fref x-%data%
                        (1 2)
                        ((1 lda) (1 *)))
          x-%offset%))))))
(t
  (setf (f2cl-lib:fref crv (1) ((1 4)))
        (-
          (* ca
             (f2cl-lib:fref a-%data%
                           (1 1)
                           ((1 lda) (1 *)))
             a-%offset%))
          (* wr d1)))
  (setf (f2cl-lib:fref crv (4) ((1 4)))
        (-
          (* ca
             (f2cl-lib:fref a-%data%
                           (2 2)
                           ((1 lda) (1 *)))
             a-%offset%))
          (* wr d2)))
  (cond
    (ltrans
      (setf (f2cl-lib:fref crv (3) ((1 4)))
            (* ca
               (f2cl-lib:fref a-%data%
                             (2 1)
                             ((1 lda) (1 *)))
               a-%offset%)))
      (setf (f2cl-lib:fref crv (2) ((1 4)))
            (* ca
               (f2cl-lib:fref a-%data%
                             (1 2)
                             ((1 lda) (1 *)))
               a-%offset%))))))
  (t
    (setf (f2cl-lib:fref crv (2) ((1 4)))

```



```

(* ca
  (f2cl-lib:fref a-%data%
                (2 1)
                ((1 lda) (1 *))
                a-%offset%)))
(setf (f2cl-lib:fref crv (3) ((1 4)))
      (* ca
        (f2cl-lib:fref a-%data%
                        (1 2)
                        ((1 lda) (1 *))
                        a-%offset%))))))
(cond
  ((= nw 1)
   (setf cmax zero)
   (setf icmax 0)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 (> j 4) nil)
   (tagbody
    (cond
      ((> (abs (f2cl-lib:fref crv (j) ((1 4)))) cmax)
       (setf cmax (abs (f2cl-lib:fref crv (j) ((1 4)))))
       (setf icmax j))))))
  (cond
    ((< cmax smini)
     (setf bnorm
           (max
            (abs
             (f2cl-lib:fref b-%data%
                           (1 1)
                           ((1 ldb$) (1 *))
                           b-%offset%))
            (abs
             (f2cl-lib:fref b-%data%
                           (2 1)
                           ((1 ldb$) (1 *))
                           b-%offset%))))))
     (cond
       ((and (< smini one) (> bnorm one))
        (if (> bnorm (* bignum smini))
            (setf scale (/ one bnorm))))))
     (setf temp (/ scale smini))
     (setf (f2cl-lib:fref x-%data%
                         (1 1)
                         ((1 ldx) (1 *))
                         x-%offset%)
           (* temp
             (f2cl-lib:fref b-%data%
                           (1 1)
                           ((1 ldb$) (1 *))
                           b-%offset%))))))

```

```

      (setf (f2cl-lib:fref x-%data%
                          (2 1)
                          ((1 ldb) (1 *))
                          x-%offset%))
      (* temp
        (f2cl-lib:fref b-%data%
                        (2 1)
                        ((1 ldb$) (1 *))
                        b-%offset%)))
      (setf xnorm (* temp bnorm))
      (setf info 1)
      (go end_label)))
(setf ur11 (f2cl-lib:fref crv (icmax) ((1 4))))
(setf cr21
  (f2cl-lib:fref crv
                  ((f2cl-lib:fref ipivot
                                  (2 icmax)
                                  ((1 4) (1 4))))
                  ((1 4))))
(setf ur12
  (f2cl-lib:fref crv
                  ((f2cl-lib:fref ipivot
                                  (3 icmax)
                                  ((1 4) (1 4))))
                  ((1 4))))
(setf cr22
  (f2cl-lib:fref crv
                  ((f2cl-lib:fref ipivot
                                  (4 icmax)
                                  ((1 4) (1 4))))
                  ((1 4))))
(setf ur11r (/ one ur11))
(setf lr21 (* ur11r cr21))
(setf ur22 (- cr22 (* ur12 lr21)))
(cond
  ((< (abs ur22) smini)
   (setf ur22 smini)
   (setf info 1)))
(cond
  ((f2cl-lib:fref rswap (icmax) ((1 4)))
   (setf br1
     (f2cl-lib:fref b-%data%
                     (2 1)
                     ((1 ldb$) (1 *))
                     b-%offset%))
   (setf br2
     (f2cl-lib:fref b-%data%
                     (1 1)
                     ((1 ldb$) (1 *))
                     b-%offset%)))

```

```

(t
  (setf br1
    (f2cl-lib:fref b-%data%
      (1 1)
      ((1 ldb$) (1 *))
      b-%offset%))

  (setf br2
    (f2cl-lib:fref b-%data%
      (2 1)
      ((1 ldb$) (1 *))
      b-%offset%))))

(setf br2 (- br2 (* lr21 br1)))
(setf bbnd (max (abs (* br1 (* ur22 ur11r))) (abs br2)))
(cond
  ((and (> bbnd one) (< (abs ur22) one))
    (if (>= bbnd (* bignum (abs ur22)))
      (setf scale (/ one bbnd)))))
(setf xr2 (/ (* br2 scale) ur22))
(setf xr1 (- (* scale br1 ur11r) (* xr2 (* ur11r ur12))))
(cond
  ((f2cl-lib:fref zswap icmax) ((1 4)))
  (setf (f2cl-lib:fref x-%data%
    (1 1)
    ((1 ldx) (1 *))
    x-%offset%)

    xr2)
  (setf (f2cl-lib:fref x-%data%
    (2 1)
    ((1 ldx) (1 *))
    x-%offset%)

    xr1))
(t
  (setf (f2cl-lib:fref x-%data%
    (1 1)
    ((1 ldx) (1 *))
    x-%offset%)

    xr1)
  (setf (f2cl-lib:fref x-%data%
    (2 1)
    ((1 ldx) (1 *))
    x-%offset%)

    xr2)))
(setf xnorm (max (abs xr1) (abs xr2)))
(cond
  ((and (> xnorm one) (> cmax one))
    (cond
      (> xnorm (f2cl-lib:f2cl/ bignum cmax))
      (setf temp (/ cmax bignum))
      (setf (f2cl-lib:fref x-%data%
        (1 1)

```

```

((1 ldx) (1 *))
x-%offset%)
(* temp
  (f2cl-lib:fref x-%data%
    (1 1)
    ((1 ldx) (1 *))
    x-%offset%)))
(setf (f2cl-lib:fref x-%data%
  (2 1)
  ((1 ldx) (1 *))
  x-%offset%))
(* temp
  (f2cl-lib:fref x-%data%
    (2 1)
    ((1 ldx) (1 *))
    x-%offset%)))
(setf xnorm (* temp xnorm))
(setf scale (* temp scale))))))
(t
  (setf (f2cl-lib:fref civ (1) ((1 4))) (* (- wi) d1))
  (setf (f2cl-lib:fref civ (2) ((1 4))) zero)
  (setf (f2cl-lib:fref civ (3) ((1 4))) zero)
  (setf (f2cl-lib:fref civ (4) ((1 4))) (* (- wi) d2))
  (setf cmax zero)
  (setf icmax 0)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j 4) nil)
  (tagbody
    (cond
      (>
        (+ (abs (f2cl-lib:fref crv (j) ((1 4))))
          (abs (f2cl-lib:fref civ (j) ((1 4)))))
        cmax)
      (setf cmax
        (+ (abs (f2cl-lib:fref crv (j) ((1 4))))
          (abs (f2cl-lib:fref civ (j) ((1 4)))))
        (setf icmax j))))))
  (cond
    (< cmax smini)
    (setf bnorm
      (max
        (+
          (abs
            (f2cl-lib:fref b-%data%
              (1 1)
              ((1 ldb$) (1 *))
              b-%offset%))
          (abs
            (f2cl-lib:fref b-%data%
              (1 2)

```

```

((1 ldb$) (1 *))
b-%offset%)))

(+
  (abs
    (f2c1-lib:fref b-%data%
      (2 1)
      ((1 ldb$) (1 *))
      b-%offset%))
    (abs
      (f2c1-lib:fref b-%data%
        (2 2)
        ((1 ldb$) (1 *))
        b-%offset%))))))
(cond
  ((and (< smini one) (> bnorm one))
    (if (> bnorm (* bignum smini))
      (setf scale (/ one bnorm))))))
(setf temp (/ scale smini))
(setf (f2c1-lib:fref x-%data%
  (1 1)
  ((1 ldx) (1 *))
  x-%offset%))
(* temp
  (f2c1-lib:fref b-%data%
    (1 1)
    ((1 ldb$) (1 *))
    b-%offset%)))
(setf (f2c1-lib:fref x-%data%
  (2 1)
  ((1 ldx) (1 *))
  x-%offset%))
(* temp
  (f2c1-lib:fref b-%data%
    (2 1)
    ((1 ldb$) (1 *))
    b-%offset%)))
(setf (f2c1-lib:fref x-%data%
  (1 2)
  ((1 ldx) (1 *))
  x-%offset%))
(* temp
  (f2c1-lib:fref b-%data%
    (1 2)
    ((1 ldb$) (1 *))
    b-%offset%)))
(setf (f2c1-lib:fref x-%data%
  (2 2)
  ((1 ldx) (1 *))
  x-%offset%))
(* temp

```

```

(f2cl-lib:fref b-%data%
  (2 2)
  ((1 ldb$) (1 *))
  b-%offset%)))
(setf xnorm (* temp bnorm))
(setf info 1)
(go end_label)))
(setf ur11 (f2cl-lib:fref crv (icmax) ((1 4))))
(setf ui11 (f2cl-lib:fref civ (icmax) ((1 4))))
(setf cr21
  (f2cl-lib:fref crv
    ((f2cl-lib:fref ipivot
      (2 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(setf ci21
  (f2cl-lib:fref civ
    ((f2cl-lib:fref ipivot
      (2 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(setf ur12
  (f2cl-lib:fref crv
    ((f2cl-lib:fref ipivot
      (3 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(setf ui12
  (f2cl-lib:fref civ
    ((f2cl-lib:fref ipivot
      (3 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(setf cr22
  (f2cl-lib:fref crv
    ((f2cl-lib:fref ipivot
      (4 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(setf ci22
  (f2cl-lib:fref civ
    ((f2cl-lib:fref ipivot
      (4 icmax)
      ((1 4) (1 4))))
    ((1 4))))
(cond
  ((or (= icmax 1) (= icmax 4))
    (cond
      (> (abs ur11) (abs ui11))
      (setf temp (/ ui11 ur11))

```

```

      (setf ur11r (/ one (* ur11 (+ one (expt temp 2)))))
      (setf ui11r (* (- temp) ur11r)))
    (t
      (setf temp (/ ur11 ui11))
      (setf ui11r (/ (- one) (* ui11 (+ one (expt temp 2)))))
      (setf ur11r (* (- temp) ui11r)))
    (setf lr21 (* cr21 ur11r))
    (setf li21 (* cr21 ui11r))
    (setf ur12s (* ur12 ur11r))
    (setf ui12s (* ur12 ui11r))
    (setf ur22 (- cr22 (* ur12 lr21)))
    (setf ui22 (- ci22 (* ur12 li21)))
    (t
      (setf ur11r (/ one ur11))
      (setf ui11r zero)
      (setf lr21 (* cr21 ur11r))
      (setf li21 (* ci21 ur11r))
      (setf ur12s (* ur12 ur11r))
      (setf ui12s (* ur12 ui11r))
      (setf ur22 (+ (- cr22 (* ur12 lr21)) (* ui12 li21)))
      (setf ui22 (- (* (- ur12) li21) (* ui12 lr21))))
    (setf u22abs (+ (abs ur22) (abs ui22)))
    (cond
      ((< u22abs smini)
       (setf ur22 smini)
       (setf ui22 zero)
       (setf info 1)))
    (cond
      ((f2cl-lib:fref rswap (icmax) ((1 4)))
       (setf br2
         (f2cl-lib:fref b-%data%
                        (1 1)
                        ((1 ldb$) (1 *))
                        b-%offset%))
       (setf br1
         (f2cl-lib:fref b-%data%
                        (2 1)
                        ((1 ldb$) (1 *))
                        b-%offset%))
       (setf bi2
         (f2cl-lib:fref b-%data%
                        (1 2)
                        ((1 ldb$) (1 *))
                        b-%offset%))
       (setf bi1
         (f2cl-lib:fref b-%data%
                        (2 2)
                        ((1 ldb$) (1 *))
                        b-%offset%)))
    (t

```

```

(setf br1
  (f2cl-lib:fref b-%data%
    (1 1)
    ((1 ldb$) (1 *))
    b-%offset%))

(setf br2
  (f2cl-lib:fref b-%data%
    (2 1)
    ((1 ldb$) (1 *))
    b-%offset%))

(setf bi1
  (f2cl-lib:fref b-%data%
    (1 2)
    ((1 ldb$) (1 *))
    b-%offset%))

(setf bi2
  (f2cl-lib:fref b-%data%
    (2 2)
    ((1 ldb$) (1 *))
    b-%offset%)))

(setf br2 (+ (- br2 (* lr21 br1)) (* li21 bi1)))
(setf bi2 (- bi2 (* li21 br1) (* lr21 bi1)))
(setf bbnd
  (max
    (* (+ (abs br1) (abs bi1))
      (* u22abs (+ (abs ur11r) (abs ui11r)))))
    (+ (abs br2) (abs bi2))))

(cond
  ((and (> bbnd one) (< u22abs one))
    (cond
      ((>= bbnd (* bignum u22abs))
        (setf scale (/ one bbnd))
        (setf br1 (* scale br1))
        (setf bi1 (* scale bi1))
        (setf br2 (* scale br2))
        (setf bi2 (* scale bi2))))))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
    (dladiv br2 bi2 ur22 ui22 xr2 xi2)
    (declare (ignore var-0 var-1 var-2 var-3))
    (setf xr2 var-4)
    (setf xi2 var-5))
  (setf xrl
    (+ (- (* ur11r br1) (* ui11r bi1) (* ur12s xr2))
      (* ui12s xi2)))
  (setf xil
    (- (+ (* ui11r br1) (* ur11r bi1))
      (* ui12s xr2)
      (* ur12s xi2)))
  (cond
    ((f2cl-lib:fref zswap (icmax) ((1 4)))

```



```

      (setf (f2cl-lib:fref x-%data%
                          (1 1)
                          ((1 ldx) (1 *)))
            x-%offset%)
      xr2)
      (setf (f2cl-lib:fref x-%data%
                          (2 1)
                          ((1 ldx) (1 *)))
            x-%offset%)
      xr1)
      (setf (f2cl-lib:fref x-%data%
                          (1 2)
                          ((1 ldx) (1 *)))
            x-%offset%)
      xi2)
      (setf (f2cl-lib:fref x-%data%
                          (2 2)
                          ((1 ldx) (1 *)))
            x-%offset%)
      xi1))
(t
  (setf (f2cl-lib:fref x-%data%
                      (1 1)
                      ((1 ldx) (1 *)))
        x-%offset%)
      xr1)
  (setf (f2cl-lib:fref x-%data%
                      (2 1)
                      ((1 ldx) (1 *)))
        x-%offset%)
      xr2)
  (setf (f2cl-lib:fref x-%data%
                      (1 2)
                      ((1 ldx) (1 *)))
        x-%offset%)
      xi1)
  (setf (f2cl-lib:fref x-%data%
                      (2 2)
                      ((1 ldx) (1 *)))
        x-%offset%)
      xi2)))
(setf xnorm
      (max (+ (abs xr1) (abs xi1)) (+ (abs xr2) (abs xi2))))
(cond
  ((and (> xnorm one) (> cmax one))
   (cond
     ((> xnorm (f2cl-lib:f2cl/ bignum cmax))
      (setf temp (/ cmax bignum))
      (setf (f2cl-lib:fref x-%data%
                          (1 1)
                          ((1 ldx) (1 *)))
            temp))))

```

[illegible]

```

nil
nil
nil
scale
xnorm
info))))))

```

dlamch LAPACK

— dlamch.input —

```

)set break resume
)sys rm -f dlamch.output
)spool dlamch.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamch.help —

```

=====
dlamch examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLAMCH - determine double precision machine parameters

SYNOPSIS

DOUBLE PRECISION FUNCTION DLAMCH(CMACH)

CHARACTER CMACH

Purpose

=====

DLAMCH determines double precision machine parameters.

Arguments

=====

CMACH (input) CHARACTER*1
 Specifies the value to be returned by DLAMCH:

| | |
|---------------|--------------------|
| = 'E' or 'e', | DLAMCH := eps |
| = 'S' or 's', | DLAMCH := sfmin |
| = 'B' or 'b', | DLAMCH := base |
| = 'P' or 'p', | DLAMCH := eps*base |
| = 'N' or 'n', | DLAMCH := t |
| = 'R' or 'r', | DLAMCH := rnd |
| = 'M' or 'm', | DLAMCH := emin |
| = 'U' or 'u', | DLAMCH := rmin |
| = 'L' or 'l', | DLAMCH := emax |
| = 'O' or 'o', | DLAMCH := rmax |

where

eps = relative machine precision
 sfmin = safe minimum, such that 1/sfmin does not overflow
 base = base of the machine
 prec = eps*base
 t = number of (base) digits in the mantissa
 rnd = 1.0 when rounding occurs in addition, 0.0 otherwise
 emin = minimum exponent before (gradual) underflow
 rmin = underflow threshold - base** (emin-1)
 emax = largest exponent before overflow
 rmax = overflow threshold - (base**emax)*(1-eps)

— dlamch.f —

```

DOUBLE PRECISION FUNCTION DLAMCH( CMACH )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1992
*
*    .. Scalar Arguments ..
*      CHARACTER          CMACH
*    ..
*
*  =====

```

```

*
*   .. Parameters ..
DOUBLE PRECISION    ONE, ZERO
PARAMETER            ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*   ..
*   .. Local Scalars ..
LOGICAL              FIRST, LRND
INTEGER              BETA, IMAX, IMIN, IT
DOUBLE PRECISION     BASE, EMAX, EMIN, EPS, PREC, RMACH, RMAX, RMIN,
$                   RND, SFMIN, SMALL, T
*
*   ..
*   .. External Functions ..
LOGICAL              LSAME
EXTERNAL             LSAME
*
*   ..
*   .. External Subroutines ..
EXTERNAL             DLAMC2
*
*   ..
*   .. Save statement ..
SAVE                FIRST, EPS, SFMIN, BASE, T, RND, EMIN, RMIN,
$                   EMAX, RMAX, PREC
*
*   ..
*   .. Data statements ..
DATA                FIRST / .TRUE. /
*
*   ..
*   .. Executable Statements ..
*
IF( FIRST ) THEN
    FIRST = .FALSE.
    CALL DLAMC2( BETA, IT, LRND, EPS, IMIN, RMIN, IMAX, RMAX )
    BASE = BETA
    T = IT
    IF( LRND ) THEN
        RND = ONE
        EPS = ( BASE**( 1-IT ) ) / 2
    ELSE
        RND = ZERO
        EPS = BASE**( 1-IT )
    END IF
    PREC = EPS*BASE
    EMIN = IMIN
    EMAX = IMAX
    SFMIN = RMIN
    SMALL = ONE / RMAX
    IF( SMALL.GE.SFMIN ) THEN
*
*       Use SMALL plus a bit, to avoid the possibility of rounding
*       causing overflow when computing 1/sfmin.
*
        SFMIN = SMALL*( ONE+EPS )

```

```

        END IF
    END IF
*
    IF( LSAME( CMACH, 'E' ) ) THEN
        RMACH = EPS
    ELSE IF( LSAME( CMACH, 'S' ) ) THEN
        RMACH = SFMIN
    ELSE IF( LSAME( CMACH, 'B' ) ) THEN
        RMACH = BASE
    ELSE IF( LSAME( CMACH, 'P' ) ) THEN
        RMACH = PREC
    ELSE IF( LSAME( CMACH, 'N' ) ) THEN
        RMACH = T
    ELSE IF( LSAME( CMACH, 'R' ) ) THEN
        RMACH = RND
    ELSE IF( LSAME( CMACH, 'M' ) ) THEN
        RMACH = EMIN
    ELSE IF( LSAME( CMACH, 'U' ) ) THEN
        RMACH = RMIN
    ELSE IF( LSAME( CMACH, 'L' ) ) THEN
        RMACH = EMAX
    ELSE IF( LSAME( CMACH, 'O' ) ) THEN
        RMACH = RMAX
    END IF
*
    DLAMCH = RMACH
    RETURN
*
*   End of DLAMCH
*
    END

```

— LAPACK dlamch —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (let ((eps 0.0)
        (sfmin 0.0)
        (base 0.0)
        (t$ 0.0f0)
        (rnd 0.0)
        (emin 0.0)
        (rmin 0.0)
        (emax 0.0)
        (rmax 0.0))

```

```

    (prec 0.0)
    (first$ nil))
  (declare (type (member t nil) first$)
            (type (single-float) t$)
            (type (double-float) prec rmax emax rmin emin rnd base sfmin eps))
  (setq first$ t)
  (defun dlamch (cmach)
    (declare (type character cmach))
    (f2cl-lib:with-multi-array-data
      ((cmach character cmach-%data% cmach-%offset%))
      (prog ((rmach 0.0) (small 0.0) (t$ 0.0) (beta 0) (imax 0) (imin 0)
             (it 0) (lrnd nil) (dlamch 0.0))
        (declare (type fixnum beta imax imin it)
                  (type (member t nil) lrnd)
                  (type (double-float) rmach small t$ dlamch))
        (cond
         (first$
          (setf first$ nil)
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
            (dlamc2 beta it lrnd eps imin rmin imax rmax)
            (declare (ignore))
            (setf beta var-0)
            (setf it var-1)
            (setf lrnd var-2)
            (setf eps var-3)
            (setf imin var-4)
            (setf rmin var-5)
            (setf imax var-6)
            (setf rmax var-7))
            (setf base (coerce (the fixnum beta) 'double-float))
            (setf t$ (coerce (the fixnum it) 'double-float))
            (cond
             (lrnd
              (setf rnd one)
              (setf eps (/ (expt base (f2cl-lib:int-sub 1 it)) 2)))
             (t
              (setf rnd zero)
              (setf eps (expt base (f2cl-lib:int-sub 1 it)))))
            (setf prec (* eps base))
            (setf emin (coerce (the fixnum imin) 'double-float))
            (setf emax (coerce (the fixnum imax) 'double-float))
            (setf sfmin rmin)
            (setf small (/ one rmax))
            (cond
             ((>= small sfmin)
              (setf sfmin (* small (+ one eps))))))
          (cond
           ((char-equal cmach #\E)
            (setf rmach eps))

```

```

      ((char-equal cmach #\S)
       (setf rmach sfmin))
      ((char-equal cmach #\B)
       (setf rmach base))
      ((char-equal cmach #\P)
       (setf rmach prec))
      ((char-equal cmach #\N)
       (setf rmach t$))
      ((char-equal cmach #\R)
       (setf rmach rnd))
      ((char-equal cmach #\M)
       (setf rmach emin))
      ((char-equal cmach #\U)
       (setf rmach rmin))
      ((char-equal cmach #\L)
       (setf rmach emax))
      ((char-equal cmach #\O)
       (setf rmach rmax)))
      (setf dlamch rmach)
end_label
      (return (values dlamch nil))))))

```

dlamc1 LAPACK

— dlamc1.input —

```

)set break resume
)sys rm -f dlamc1.output
)spool dlamc1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamc1.help —

```

=====
dlamc1 examples
=====

```



```
=====
Man Page Details
=====
```

```
-- LAPACK auxiliary routine (version 1.1) --
   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
   Courant Institute, Argonne National Lab, and Rice University
   October 31, 1992
```

```
.. Scalar Arguments ..
<    LOGICAL          IEEE1, RND >
<    INTEGER          BETA, T >
..
```

```
Purpose
=====
```

DLAMC1 determines the machine parameters given by BETA, T, RND, and IEEE1.

```
Arguments
=====
```

```
BETA    (output) INTEGER
         The base of the machine.

T        (output) INTEGER
         The number of ( BETA ) digits in the mantissa.

RND      (output) LOGICAL
         Specifies whether proper rounding ( RND = .TRUE. ) or
         chopping ( RND = .FALSE. ) occurs in addition. This may not
         be a reliable guide to the way in which the machine performs
         its arithmetic.

IEEE1    (output) LOGICAL
         Specifies whether rounding appears to be done in the IEEE
         'round to nearest' style.
```

```
Further Details
=====
```

See Malcolm M. A. (1972) Algorithms to reveal properties of floating-point arithmetic. Comms. of the ACM, 15, 949-951.

See Gentleman W. M. and Marovich S. B. (1974) More on algorithms that reveal properties of floating point arithmetic units. Comms. of the ACM, 17, 276-277.

— dlamc1.f —

```

*****
*
*      SUBROUTINE DLAMC1( BETA, T, RND, IEEE1 )
*
*      -- LAPACK auxiliary routine (version 3.0) --
*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      October 31, 1992
*
*      .. Scalar Arguments ..
*      LOGICAL          IEEE1, RND
*      INTEGER          BETA, T
*      ..
*
*      =====
*
*      .. Local Scalars ..
*      LOGICAL          FIRST, LIEEE1, LRND
*      INTEGER          LBETA, LT
*      DOUBLE PRECISION A, B, C, F, ONE, QTR, SAVEC, T1, T2
*      ..
*      .. External Functions ..
*      DOUBLE PRECISION DLAMC3
*      EXTERNAL          DLAMC3
*      ..
*      .. Save statement ..
*      SAVE              FIRST, LIEEE1, LBETA, LRND, LT
*      ..
*      .. Data statements ..
*      DATA              FIRST / .TRUE. /
*      ..
*      .. Executable Statements ..
*
*      IF( FIRST ) THEN
*          FIRST = .FALSE.
*          ONE = 1
*
*      LBETA, LIEEE1, LT and LRND are the local values of BETA,
*      IEEE1, T and RND.
*
*      Throughout this routine we use the function DLAMC3 to ensure
*      that relevant values are stored and not held in registers, or
*      are not affected by optimizers.
*
*      Compute a = 2.0**m with the smallest positive integer m such

```

```

*      that
*
*      fl( a + 1.0 ) = a.
*
*      A = 1
*      C = 1
*
**+   WHILE( C.EQ.ONE )LOOP
10    CONTINUE
      IF( C.EQ.ONE ) THEN
        A = 2*A
        C = DLAMC3( A, ONE )
        C = DLAMC3( C, -A )
        GO TO 10
      END IF
**+   END WHILE
*
*      Now compute b = 2.0**m with the smallest positive integer m
*      such that
*
*      fl( a + b ) .gt. a.
*
*      B = 1
*      C = DLAMC3( A, B )
*
**+   WHILE( C.EQ.A )LOOP
20    CONTINUE
      IF( C.EQ.A ) THEN
        B = 2*B
        C = DLAMC3( A, B )
        GO TO 20
      END IF
**+   END WHILE
*
*      Now compute the base. a and c are neighbouring floating point
*      numbers in the interval ( beta**t, beta**( t + 1 ) ) and so
*      their difference is beta. Adding 0.25 to c is to ensure that it
*      is truncated to beta and not ( beta - 1 ).
*
*      QTR = ONE / 4
*      SAVEC = C
*      C = DLAMC3( C, -A )
*      LBETA = C + QTR
*
*      Now determine whether rounding or chopping occurs, by adding a
*      bit less than beta/2 and a bit more than beta/2 to a.
*
*      B = LBETA
*      F = DLAMC3( B / 2, -B / 100 )
*      C = DLAMC3( F, A )

```

```

      IF( C.EQ.A ) THEN
        LRND = .TRUE.
      ELSE
        LRND = .FALSE.
      END IF
      F = DLAMC3( B / 2, B / 100 )
      C = DLAMC3( F, A )
      IF( ( LRND ) .AND. ( C.EQ.A ) )
$      LRND = .FALSE.
*
*      Try and decide whether rounding is done in the IEEE 'round to
*      nearest' style. B/2 is half a unit in the last place of the two
*      numbers A and SAVEC. Furthermore, A is even, i.e. has last bit
*      zero, and SAVEC is odd. Thus adding B/2 to A should not change
*      A, but adding B/2 to SAVEC should change SAVEC.
*
      T1 = DLAMC3( B / 2, A )
      T2 = DLAMC3( B / 2, SAVEC )
      LIEEE1 = ( T1.EQ.A ) .AND. ( T2.GT.SAVEC ) .AND. LRND
*
*      Now find the mantissa, t. It should be the integer part of
*      log to the base beta of a, however it is safer to determine t
*      by powering. So we find t as the smallest positive integer for
*      which
*
      fl( beta**t + 1.0 ) = 1.0.
*
      LT = 0
      A = 1
      C = 1
*
*+      WHILE( C.EQ.ONE )LOOP
30      CONTINUE
      IF( C.EQ.ONE ) THEN
        LT = LT + 1
        A = A*LBETA
        C = DLAMC3( A, ONE )
        C = DLAMC3( C, -A )
        GO TO 30
      END IF
*+      END WHILE
*
      END IF
*
      BETA = LBETA
      T = LT
      RND = LRND
      IEEE1 = LIEEE1
      RETURN
*

```

```

*      End of DLAMC1
*
      END
*

```

— LAPACK dlamc1 —

```

(let ((lieee1 nil) (lbeta 0) (lrnd nil) (f2cl-lib:lt 0) (first$ nil))
  (declare (type fixnum f2cl-lib:lt lbeta)
            (type (member t nil) first$ lrnd lieee1))
  (setq first$ t)
  (defun dlamc1 (beta t$ rnd ieeee1)
    (declare (type (member t nil) ieeee1 rnd)
              (type fixnum t$ beta))
    (prog ((a 0.0) (b 0.0) (c 0.0) (f 0.0) (one 0.0) (qtr 0.0) (savec 0.0)
           (t1 0.0) (t2 0.0))
      (declare (type (double-float) t2 t1 savec qtr one f c b a))
      (cond
        (first$
         (tagbody
          (setf first$ nil)
          (setf one (coerce (the fixnum 1) 'double-float))
          (setf a (coerce (the fixnum 1) 'double-float))
          (setf c (coerce (the fixnum 1) 'double-float))
          label10
          (cond
            ((= c one)
             (setf a (* 2 a))
             (setf c
                  (multiple-value-bind (ret-val var-0 var-1)
                    (dlamc3 a one)
                    (declare (ignore))
                    (setf a var-0)
                    (setf one var-1)
                    ret-val))
             (setf c
                  (multiple-value-bind (ret-val var-0 var-1)
                    (dlamc3 c (- a))
                    (declare (ignore var-1))
                    (setf c var-0)
                    ret-val))
             (go label10)))
          (setf b (coerce (the fixnum 1) 'double-float))
          (setf c
                (multiple-value-bind (ret-val var-0 var-1)
                  (dlamc3 a b)

```

```

                                (declare (ignore))
                                (setf a var-0)
                                (setf b var-1)
                                ret-val))
label20
(cond
  ((= c a)
    (setf b (* 2 b))
    (setf c
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 a b)
        (declare (ignore))
        (setf a var-0)
        (setf b var-1)
        ret-val)))
    (go label20)))
(setf qtr (/ one 4))
(setf savec c)
(setf c
  (multiple-value-bind (ret-val var-0 var-1)
    (dlamc3 c (- a))
    (declare (ignore var-1))
    (setf c var-0)
    ret-val))
  (setf lbeta (f2cl-lib:int (+ c qtr)))
  (setf b (coerce (the fixnum lbeta) 'double-float))
  (setf f (dlamc3 (/ b 2) (/ (- b) 100)))
  (setf c
    (multiple-value-bind (ret-val var-0 var-1)
      (dlamc3 f a)
      (declare (ignore))
      (setf f var-0)
      (setf a var-1)
      ret-val))
    (cond
      ((= c a)
        (setf lrnd t))
      (t
        (setf lrnd nil)))
    (setf f (dlamc3 (/ b 2) (/ b 100)))
    (setf c
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 f a)
        (declare (ignore))
        (setf f var-0)
        (setf a var-1)
        ret-val))
      (if (and lrnd (= c a)) (setf lrnd nil))
      (setf t1
        (multiple-value-bind (ret-val var-0 var-1)

```

```

        (dlamc3 (/ b 2) a)
        (declare (ignore var-0))
        (setf a var-1)
        ret-val))
    (setf t2
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 (/ b 2) savec)
        (declare (ignore var-0))
        (setf savec var-1)
        ret-val))
    (setf lieee1 (and (= t1 a) (> t2 savec) lrnd))
    (setf f2cl-lib:lt 0)
    (setf a (coerce (the fixnum 1) 'double-float))
    (setf c (coerce (the fixnum 1) 'double-float))
label30
    (cond
      ((= c one)
        (setf f2cl-lib:lt (f2cl-lib:int-add f2cl-lib:lt 1))
        (setf a (* a lbeta))
        (setf c
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 a one)
            (declare (ignore))
            (setf a var-0)
            (setf one var-1)
            ret-val))
          (setf c
            (multiple-value-bind (ret-val var-0 var-1)
              (dlamc3 c (- a))
              (declare (ignore var-1))
              (setf c var-0)
              ret-val))
            (go label30))))))
    (setf beta lbeta)
    (setf t$ f2cl-lib:lt)
    (setf rnd lrnd)
    (setf ieee1 lieee1)
end_label
    (return (values beta t$ rnd ieee1))))

```

dlamc2 LAPACK

— dlamc2.input —

```

)set break resume
)sys rm -f dlamc2.output
)spool dlamc2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamc2.help —

```

=====
dlamc2 examples
=====

```

```

=====
Man Page Details
=====

```

```

-- LAPACK auxiliary routine (version 1.1) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992

```

```

.. Scalar Arguments ..
<    LOGICAL          RND >
<    INTEGER          BETA, EMAX, EMIN, T >
<    DOUBLE PRECISION EPS, RMAX, RMIN >
..

```

```

Purpose
=====

```

DLAMC2 determines the machine parameters specified in its argument list.

```

Arguments
=====

```

```

BETA    (output) INTEGER
         The base of the machine.

T       (output) INTEGER
         The number of ( BETA ) digits in the mantissa.

RND     (output) LOGICAL

```


Specifies whether proper rounding (RND = .TRUE.) or chopping (RND = .FALSE.) occurs in addition. This may not be a reliable guide to the way in which the machine performs its arithmetic.

EPS (output) DOUBLE PRECISION
The smallest positive number such that

$$fl(1.0 - EPS) < 1.0,$$

where fl denotes the computed value.

EMIN (output) INTEGER
The minimum exponent before (gradual) underflow occurs.

RMIN (output) DOUBLE PRECISION
The smallest normalized number for the machine, given by $BASE^{EMIN-1}$, where BASE is the floating point value of BETA.

EMAX (output) INTEGER
The maximum exponent before overflow occurs.

RMAX (output) DOUBLE PRECISION
The largest positive number for the machine, given by $BASE^{EMAX} * (1 - EPS)$, where BASE is the floating point value of BETA.

Further Details
=====

The computation of EPS is based on a routine PARANOIA by W. Kahan of the University of California at Berkeley.

— dlamc2.f —

```
*****
*
*      SUBROUTINE DLAMC2( BETA, T, RND, EPS, EMIN, RMIN, EMAX, RMAX )
*
*      -- LAPACK auxiliary routine (version 3.0) --
*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      October 31, 1992
*
*      .. Scalar Arguments ..
```

```

LOGICAL          RND
INTEGER          BETA, EMAX, EMIN, T
DOUBLE PRECISION EPS, RMAX, RMIN
*
* ..
*
* =====
*
* .. Local Scalars ..
LOGICAL          FIRST, IEEE, IWARN, LIEEE1, LRND
INTEGER          GNMIN, GPMIN, I, LBETA, LEMAX, LEMIN, LT,
$ NGNMIN, NGPMIN
DOUBLE PRECISION A, B, C, HALF, LEPS, LRMAX, LRMIN, ONE, RBASE,
$ SIXTH, SMALL, THIRD, TWO, ZERO
*
* ..
* .. External Functions ..
DOUBLE PRECISION DLAMC3
EXTERNAL         DLAMC3
*
* ..
* .. External Subroutines ..
EXTERNAL         DLAMC1, DLAMC4, DLAMC5
*
* ..
* .. Intrinsic Functions ..
INTRINSIC        ABS, MAX, MIN
*
* ..
* .. Save statement ..
SAVE            FIRST, IWARN, LBETA, LEMAX, LEMIN, LEPS, LRMAX,
$ LRMIN, LT
*
* ..
* .. Data statements ..
DATA            FIRST / .TRUE. / , IWARN / .FALSE. /
*
* ..
* .. Executable Statements ..
*
IF( FIRST ) THEN
    FIRST = .FALSE.
    ZERO = 0
    ONE = 1
    TWO = 2
*
* LBETA, LT, LRND, LEPS, LEMIN and LRMIN are the local values of
* BETA, T, RND, EPS, EMIN and RMIN.
*
* Throughout this routine we use the function DLAMC3 to ensure
* that relevant values are stored and not held in registers, or
* are not affected by optimizers.
*
* DLAMC1 returns the parameters LBETA, LT, LRND and LIEEE1.
*
CALL DLAMC1( LBETA, LT, LRND, LIEEE1 )
*

```

```

*      Start to find EPS.
*
      B = LBETA
      A = B**( -LT )
      LEPS = A
*
*      Try some tricks to see whether or not this is the correct  EPS.
*
      B = TWO / 3
      HALF = ONE / 2
      SIXTH = DLAMC3( B, -HALF )
      THIRD = DLAMC3( SIXTH, SIXTH )
      B = DLAMC3( THIRD, -HALF )
      B = DLAMC3( B, SIXTH )
      B = ABS( B )
      IF( B.LT.LEPS )
$        B = LEPS
*
      LEPS = 1
*
*+
10  CONTINUE
      IF( ( LEPS.GT.B ) .AND. ( B.GT.ZERO ) ) THEN
          LEPS = B
          C = DLAMC3( HALF*LEPS, ( TWO**5 )*( LEPS**2 ) )
          C = DLAMC3( HALF, -C )
          B = DLAMC3( HALF, C )
          C = DLAMC3( HALF, -B )
          B = DLAMC3( HALF, C )
          GO TO 10
      END IF
*+
      END WHILE
*
      IF( A.LT.LEPS )
$        LEPS = A
*
*      Computation of EPS complete.
*
*      Now find EMIN. Let A = + or - 1, and + or - (1 + BASE**(-3)).
*      Keep dividing A by BETA until (gradual) underflow occurs. This
*      is detected when we cannot recover the previous A.
*
      RBASE = ONE / LBETA
      SMALL = ONE
      DO 20 I = 1, 3
          SMALL = DLAMC3( SMALL*RBASE, ZERO )
20  CONTINUE
      A = DLAMC3( ONE, SMALL )
      CALL DLAMC4( NGPMIN, ONE, LBETA )
      CALL DLAMC4( NGNMIN, -ONE, LBETA )

```

```

CALL DLAMC4( GPMIN, A, LBETA )
CALL DLAMC4( GNMIN, -A, LBETA )
IEEE = .FALSE.

*
IF( ( NGPMIN.EQ.NGNMIN ) .AND. ( GPMIN.EQ.GNMIN ) ) THEN
  IF( NGPMIN.EQ.GPMIN ) THEN
    LEMIN = NGPMIN
  *   ( Non twos-complement machines, no gradual underflow;
  *   e.g., VAX )
  ELSE IF( ( GPMIN-NGPMIN ).EQ.3 ) THEN
    LEMIN = NGPMIN - 1 + LT
    IEEE = .TRUE.
  *   ( Non twos-complement machines, with gradual underflow;
  *   e.g., IEEE standard followers )
  ELSE
    LEMIN = MIN( NGPMIN, GPMIN )
  *   ( A guess; no known machine )
    IWARN = .TRUE.
  END IF

*
ELSE IF( ( NGPMIN.EQ.GPMIN ) .AND. ( NGNMIN.EQ.GNMIN ) ) THEN
  IF( ABS( NGPMIN-NGNMIN ).EQ.1 ) THEN
    LEMIN = MAX( NGPMIN, NGNMIN )
  *   ( Twos-complement machines, no gradual underflow;
  *   e.g., CYBER 205 )
  ELSE
    LEMIN = MIN( NGPMIN, NGNMIN )
  *   ( A guess; no known machine )
    IWARN = .TRUE.
  END IF

*
ELSE IF( ( ABS( NGPMIN-NGNMIN ).EQ.1 ) .AND.
$   ( GPMIN.EQ.GNMIN ) ) THEN
  IF( ( GPMIN-MIN( NGPMIN, NGNMIN ) ).EQ.3 ) THEN
    LEMIN = MAX( NGPMIN, NGNMIN ) - 1 + LT
  *   ( Twos-complement machines with gradual underflow;
  *   no known machine )
  ELSE
    LEMIN = MIN( NGPMIN, NGNMIN )
  *   ( A guess; no known machine )
    IWARN = .TRUE.
  END IF

*
ELSE
  LEMIN = MIN( NGPMIN, NGNMIN, GPMIN, GNMIN )
  *   ( A guess; no known machine )
  IWARN = .TRUE.
END IF

***
* Comment out this if block if EMIN is ok

```

```

      IF( IWARN ) THEN
        FIRST = .TRUE.
        WRITE( 6, FMT = 9999 )LEMIN
      END IF
***
*
*   Assume IEEE arithmetic if we found denormalised numbers above,
*   or if arithmetic seems to round in the IEEE style, determined
*   in routine DLAMC1. A true IEEE machine should have both things
*   true; however, faulty machines may have one or the other.
*
      IEEE = IEEE .OR. LIEEE1
*
*   Compute RMIN by successive division by BETA. We could compute
*   RMIN as BASE**( EMIN - 1 ), but some machines underflow during
*   this computation.
*
      LRMIN = 1
      DO 30 I = 1, 1 - LEMIN
        LRMIN = DLAMC3( LRMIN*RBASE, ZERO )
30    CONTINUE
*
*   Finally, call DLAMC5 to compute EMAX and RMAX.
*
      CALL DLAMC5( LBETA, LT, LEMIN, IEEE, LEMAX, LRMAX )
      END IF
*
      BETA = LBETA
      T = LT
      RND = LRND
      EPS = LEPS
      EMIN = LEMIN
      RMIN = LRMIN
      EMAX = LEMAX
      RMAX = LRMAX
*
      RETURN
*
9999 FORMAT( / / ' WARNING. The value EMIN may be incorrect:-',
$          ' EMIN = ', I8, /
$          ' If, after inspection, the value EMIN looks',
$          ' acceptable please comment out ',
$          / ' the IF block as marked within the code of routine',
$          ' DLAMC2,', / ' otherwise supply EMIN explicitly.', / )
*
*   End of DLAMC2
*
      END

```

— LAPACK dlamc2 —

```

(let ((lbeta 0)
      (lmax 0)
      (lmin 0)
      (leps 0.0)
      (lrmax 0.0)
      (lrmin 0.0)
      (f2cl-lib:lt 0)
      (first$ nil)
      (iwarn nil))
  (declare (type (member t nil) iwarn first$)
           (type (double-float) lrmin lrmax leps)
           (type fixnum f2cl-lib:lt lmin lmax lbeta))
  (setq first$ t)
  (setq iwarn nil)
  (defun dlamc2 (beta t$ rnd eps emin rmin emax rmax)
    (declare (type (double-float) rmax rmin eps)
             (type (member t nil) rnd)
             (type fixnum emax emin t$ beta))
    (prog ((a 0.0) (b 0.0) (c 0.0) (half 0.0) (one 0.0) (rbase 0.0)
           (sixth$ 0.0) (small 0.0) (third$ 0.0) (two 0.0) (zero 0.0) (gnmin 0)
           (gpmin 0) (i 0) (ngnmin 0) (ngpmin 0) (ieee nil) (lieee1 nil)
           (lrnd nil))
      (declare (type (member t nil) lrnd lieee1 ieee)
               (type fixnum ngpmin ngnmin i gpmin gnmin)
               (type (double-float) zero two third$ small sixth$ rbase one half
                      c b a))
      (cond
        (first$
         (tagbody
          (setf first$ nil)
          (setf zero (coerce (the fixnum 0) 'double-float))
          (setf one (coerce (the fixnum 1) 'double-float))
          (setf two (coerce (the fixnum 2) 'double-float))
          (multiple-value-bind (var-0 var-1 var-2 var-3)
            (dlamc1 lbeta f2cl-lib:lt lrnd lieee1)
            (declare (ignore))
            (setf lbeta var-0)
            (setf f2cl-lib:lt var-1)
            (setf lrnd var-2)
            (setf lieee1 var-3))
          (setf b (coerce (the fixnum lbeta) 'double-float))
          (setf a (expt b (f2cl-lib:int-sub f2cl-lib:lt)))
          (setf leps a)
          (setf b (/ two 3))
          (setf half (/ one 2))
          (setf sixth$
            (multiple-value-bind (ret-val var-0 var-1)

```

```

        (dlamc3 b (- half))
        (declare (ignore var-1))
        (setf b var-0)
        ret-val))
(setf third$
  (multiple-value-bind (ret-val var-0 var-1)
    (dlamc3 sixth$ sixth$)
    (declare (ignore))
    (setf sixth$ var-0)
    (setf sixth$ var-1)
    ret-val))
(setf b
  (multiple-value-bind (ret-val var-0 var-1)
    (dlamc3 third$ (- half))
    (declare (ignore var-1))
    (setf third$ var-0)
    ret-val))
(setf b
  (multiple-value-bind (ret-val var-0 var-1)
    (dlamc3 b sixth$)
    (declare (ignore))
    (setf b var-0)
    (setf sixth$ var-1)
    ret-val))
(setf b (abs b))
(if (< b leps) (setf b leps))
(setf leps (coerce (the fixnum 1) 'double-float))
label10
(cond
  ((and (> leps b) (> b zero))
   (setf leps b)
   (setf c (dlamc3 (* half leps) (* (expt two 5) (expt leps 2))))
   (setf c
     (multiple-value-bind (ret-val var-0 var-1)
       (dlamc3 half (- c))
       (declare (ignore var-1))
       (setf half var-0)
       ret-val))
   (setf b
     (multiple-value-bind (ret-val var-0 var-1)
       (dlamc3 half c)
       (declare (ignore))
       (setf half var-0)
       (setf c var-1)
       ret-val))
   (setf c
     (multiple-value-bind (ret-val var-0 var-1)
       (dlamc3 half (- b))
       (declare (ignore var-1))
       (setf half var-0)

```

```

        ret-val))
    (setf b
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 half c)
        (declare (ignore))
        (setf half var-0)
        (setf c var-1)
        ret-val))
    (go label10)))
(if (< a leps) (setf leps a))
(setf rbase (/ one lbeta))
(setf small one)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i 3) nil)
  (tagbody
    (setf small
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 (* small rbase) zero)
        (declare (ignore var-0))
        (setf zero var-1)
        ret-val))))))
(setf a
  (multiple-value-bind (ret-val var-0 var-1)
    (dlamc3 one small)
    (declare (ignore))
    (setf one var-0)
    (setf small var-1)
    ret-val))
(multiple-value-bind (var-0 var-1 var-2)
  (dlamc4 ngpmin one lbeta)
  (declare (ignore var-1 var-2))
  (setf ngpmin var-0))
(multiple-value-bind (var-0 var-1 var-2)
  (dlamc4 ngnmin (- one) lbeta)
  (declare (ignore var-1 var-2))
  (setf ngnmin var-0))
(multiple-value-bind (var-0 var-1 var-2)
  (dlamc4 gpmin a lbeta)
  (declare (ignore var-1 var-2))
  (setf gpmin var-0))
(multiple-value-bind (var-0 var-1 var-2)
  (dlamc4 gnmin (- a) lbeta)
  (declare (ignore var-1 var-2))
  (setf gnmin var-0))
(setf ieee nil)
(cond
  ((and (= ngpmin ngnmin) (= gpmin gnmin))
    (cond
      ((= ngpmin gpmin)
        (setf lemin ngpmin))

```



```

(= (f2cl-lib:int-add gpmin (f2cl-lib:int-sub ngpmin)) 3)
(setf lemin
  (f2cl-lib:int-add (f2cl-lib:int-sub ngpmin 1)
    f2cl-lib:lt))

(setf ieee t))
(t
  (setf lemin
    (min (the fixnum ngpmin)
      (the fixnum gpmin)))
    (setf iwarn t))))
((and (= ngpmin gpmin) (= ngnmin gnmin))
  (cond
    ((= (abs (f2cl-lib:int-add ngpmin (f2cl-lib:int-sub ngnmin))) 1)
      (setf lemin
        (max (the fixnum ngpmin)
          (the fixnum ngnmin))))

    (t
      (setf lemin
        (min (the fixnum ngpmin)
          (the fixnum ngnmin)))
        (setf iwarn t))))))
((and
  (= (abs (f2cl-lib:int-add ngpmin (f2cl-lib:int-sub ngnmin))) 1)
  (= gpmin gnmin))
  (cond
    ((=
      (f2cl-lib:int-add gpmin
        (f2cl-lib:int-sub
          (min (the fixnum ngpmin)
            (the fixnum ngnmin))))

      3)
      (setf lemin
        (f2cl-lib:int-add
          (f2cl-lib:int-sub
            (max (the fixnum ngpmin)
              (the fixnum ngnmin))
            1)
          f2cl-lib:lt)))

    (t
      (setf lemin
        (min (the fixnum ngpmin)
          (the fixnum ngnmin)))
        (setf iwarn t))))))
(t
  (setf lemin
    (min (the fixnum ngpmin)
      (the fixnum ngnmin)
      (the fixnum gpmin)
      (the fixnum gnmin)))
    (setf iwarn t)))

```

```

(cond
  (iwarn
    (setf first$ t)
    (format t "~&~s~a~s~%~s~%~s~%"
      "WARNING. The value EMIN may be incorrect:- EMIN = "
      lemin
      " If, after inspection, the value EMIN looks acceptable "
      "please comment out "
      " the IF block as marked within the code of routine DLAMC2,"
      " otherwise supply EMIN explicitly.")))
  (setf ieee (or ieee lieee1))
  (setf lrmin (coerce (the fixnum 1) 'double-float))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i (f2cl-lib:int-add 1 (f2cl-lib:int-sub lemin)))
     nil)
    (tagbody
      (setf lrmin
        (multiple-value-bind (ret-val var-0 var-1)
          (dlamc3 (* lrmin rbase) zero)
          (declare (ignore var-0))
          (setf zero var-1)
          ret-val))))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
        (dlamc5 lbeta f2cl-lib:lt lemin ieee lemax lrmax)
        (declare (ignore var-1))
        (setf lbeta var-0)
        (setf lemin var-2)
        (setf ieee var-3)
        (setf lemax var-4)
        (setf lrmax var-5))))))
  (setf beta lbeta)
  (setf t$ f2cl-lib:lt)
  (setf rnd lrnd)
  (setf eps leps)
  (setf emin lemin)
  (setf rmin lrmin)
  (setf emax lemax)
  (setf rmax lrmax)
end_label
(return (values beta t$ rnd eps emin rmin emax rmax))))

```

dlamc3 LAPACK

— dlamc3.input —

```

)set break resume
)sys rm -f dlamc3.output
)spool dlamc3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamc3.help —

```

=====
dlamc3 examples
=====

```

```

=====
Man Page Details
=====

```

```

-- LAPACK auxiliary routine (version 1.1) --
   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
   Courant Institute, Argonne National Lab, and Rice University
   October 31, 1992

   .. Scalar Arguments ..
<      DOUBLE PRECISION   A, B >
   ..

```

```

Purpose
=====

```

DLAMC3 is intended to force A and B to be stored prior to doing the addition of A and B, for use in situations where optimizers might hold one of these in a register.

```

Arguments
=====

```

```

A, B      (input) DOUBLE PRECISION
           The values A and B.

```

— dlamc3.f —

```

*****
*
*      DOUBLE PRECISION FUNCTION DLAMC3( A, B )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1992
*
*  .. Scalar Arguments ..
*  DOUBLE PRECISION   A, B
*  ..
*
*  =====
*
*  .. Executable Statements ..
*
*      DLAMC3 = A + B
*
*      RETURN
*
*  End of DLAMC3
*
*      END
*

```

— LAPACK dlamc3 —

```

(defun dlamc3 (a b)
  (declare (type (double-float) b a))
  (prog ((dlamc3 0.0))
    (declare (type (double-float) dlamc3))
    (setf dlamc3 (+ a b))
    (return (values dlamc3 a b))))

```

dlamc4 LAPACK

— dlamc4.input —

```

)set break resume
)sys rm -f dlamc4.output
)spool dlamc4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamc4.help —

```

=====
dlamc4 examples
=====

```

```

=====
Man Page Details
=====

```

```

-- LAPACK auxiliary routine (version 1.1) --
   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
   Courant Institute, Argonne National Lab, and Rice University
   October 31, 1992

   .. Scalar Arguments ..
<      INTEGER          BASE, EMIN >
<      DOUBLE PRECISION  START >
   ..

Purpose
=====

DLAMC4 is a service routine for DLAMC2.

Arguments
=====

EMIN      (output) EMIN
          The minimum exponent before (gradual) underflow, computed by
          setting A = START and dividing by BASE until the previous A
          can not be recovered.

START     (input) DOUBLE PRECISION
          The starting point for determining EMIN.

BASE      (input) INTEGER

```

The base of the machine.

— dlamc4.f —

```
*****
*
*      SUBROUTINE DLAMC4( EMIN, START, BASE )
*
*      -- LAPACK auxiliary routine (version 3.0) --
*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      October 31, 1992
*
*      .. Scalar Arguments ..
*      INTEGER          BASE, EMIN
*      DOUBLE PRECISION START
*
*      ..
*
*      =====
*
*      .. Local Scalars ..
*      INTEGER          I
*      DOUBLE PRECISION A, B1, B2, C1, C2, D1, D2, ONE, RBASE, ZERO
*
*      ..
*      .. External Functions ..
*      DOUBLE PRECISION DLAMC3
*      EXTERNAL          DLAMC3
*
*      ..
*      .. Executable Statements ..
*
*      A = START
*      ONE = 1
*      RBASE = ONE / BASE
*      ZERO = 0
*      EMIN = 1
*      B1 = DLAMC3( A*RBASE, ZERO )
*      C1 = A
*      C2 = A
*      D1 = A
*      D2 = A
*+  WHILE( ( C1.EQ.A ).AND.( C2.EQ.A ).AND.
*  $      ( D1.EQ.A ).AND.( D2.EQ.A ) ) LOOP
10  CONTINUE
*      IF( ( C1.EQ.A ) .AND. ( C2.EQ.A ) .AND. ( D1.EQ.A ) .AND.
*        $      ( D2.EQ.A ) ) THEN
*          EMIN = EMIN - 1
```

```

      A = B1
      B1 = DLAMC3( A / BASE, ZERO )
      C1 = DLAMC3( B1*BASE, ZERO )
      D1 = ZERO
      DO 20 I = 1, BASE
        D1 = D1 + B1
20    CONTINUE
      B2 = DLAMC3( A*RBASE, ZERO )
      C2 = DLAMC3( B2 / RBASE, ZERO )
      D2 = ZERO
      DO 30 I = 1, BASE
        D2 = D2 + B2
30    CONTINUE
      GO TO 10
    END IF
  **+  END WHILE
  *
    RETURN
  *
  *    End of DLAMC4
  *
  END
  *

```

— LAPACK dlamc4 —

```

(defun dlamc4 (emin start base)
  (declare (type (double-float) start) (type fixnum base emin))
  (prog ((a 0.0) (b1 0.0) (b2 0.0) (c1 0.0) (c2 0.0) (d1 0.0) (d2 0.0)
    (one 0.0) (rbase 0.0) (zero 0.0) (i 0))
    (declare (type fixnum i)
      (type (double-float) zero rbase one d2 d1 c2 c1 b2 b1 a))
    (setf a start)
    (setf one (coerce (the fixnum 1) 'double-float))
    (setf rbase (/ one base))
    (setf zero (coerce (the fixnum 0) 'double-float))
    (setf emin 1)
    (setf b1
      (multiple-value-bind (ret-val var-0 var-1)
        (dlamc3 (* a rbase) zero)
        (declare (ignore var-0))
        (setf zero var-1)
        ret-val))
    (setf c1 a)
    (setf c2 a)
    (setf d1 a)

```

```

      (setf d2 a)
label10
      (cond
        ((and (= c1 a) (= c2 a) (= d1 a) (= d2 a))
          (setf emin (f2cl-lib:int-sub emin 1))
          (setf a b1)
          (setf b1
            (multiple-value-bind (ret-val var-0 var-1)
              (dlamc3 (/ a base) zero)
              (declare (ignore var-0))
              (setf zero var-1)
              ret-val)))
        (setf c1
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 (* b1 base) zero)
            (declare (ignore var-0))
            (setf zero var-1)
            ret-val)))
        (setf d1 zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i base) nil)
          (tagbody (setf d1 (+ d1 b1)) label20))
        (setf b2
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 (* a rbase) zero)
            (declare (ignore var-0))
            (setf zero var-1)
            ret-val)))
        (setf c2
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 (/ b2 rbase) zero)
            (declare (ignore var-0))
            (setf zero var-1)
            ret-val)))
        (setf d2 zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i base) nil)
          (tagbody (setf d2 (+ d2 b2)) label30))
        (go label10)))
end_label
      (return (values emin nil nil)))

```

dlamc5 LAPACK

— dlamc5.input —

```

)set break resume
)sys rm -f dlamc5.output
)spool dlamc5.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamc5.help —

```

=====
dlamc5 examples
=====

```

```

=====
Man Page Details
=====

```

```

-- LAPACK auxiliary routine (version 1.1) --
   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
   Courant Institute, Argonne National Lab, and Rice University
   October 31, 1992

   .. Scalar Arguments ..
<      LOGICAL          IEEE >
<      INTEGER          BETA, EMAX, EMIN, P >
<      DOUBLE PRECISION RMAX >
   ..

```

```

Purpose
=====

```

DLAMC5 attempts to compute RMAX, the largest machine floating-point number, without overflow. It assumes that $EMAX + \text{abs}(EMIN)$ sum approximately to a power of 2. It will fail on machines where this assumption does not hold, for example, the Cyber 205 ($EMIN = -28625$, $EMAX = 28718$). It will also fail if the value supplied for EMIN is too large (i.e. too close to zero), probably with overflow.

```

Arguments
=====

```

```

BETA      (input) INTEGER

```

The base of floating-point arithmetic.

P (input) INTEGER
 The number of base BETA digits in the mantissa of a
 floating-point value.

EMIN (input) INTEGER
 The minimum exponent before (gradual) underflow.

IEEE (input) LOGICAL
 A logical flag specifying whether or not the arithmetic
 system is thought to comply with the IEEE standard.

EMAX (output) INTEGER
 The largest exponent before overflow

RMAX (output) DOUBLE PRECISION
 The largest machine floating-point number.

— dlamc5.f —

```
*****
*
*      SUBROUTINE DLAMC5( BETA, P, EMIN, IEEE, EMAX, RMAX )
*
*      -- LAPACK auxiliary routine (version 3.0) --
*      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*      Courant Institute, Argonne National Lab, and Rice University
*      October 31, 1992
*
*      .. Scalar Arguments ..
*      LOGICAL          IEEE
*      INTEGER          BETA, EMAX, EMIN, P
*      DOUBLE PRECISION RMAX
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION ZERO, ONE
*      PARAMETER        ( ZERO = 0.0D0, ONE = 1.0D0 )
*      ..
*      .. Local Scalars ..
*      INTEGER          EXBITS, EXPSUM, I, LEXP, NBITS, TRY, UEXP
*      DOUBLE PRECISION OLDY, RECBAS, Y, Z
*      ..
```

```

*      .. External Functions ..
      DOUBLE PRECISION  DLAMC3
      EXTERNAL          DLAMC3
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          MOD
*
*      ..
*      .. Executable Statements ..
*
*      First compute LEXP and UEXP, two powers of 2 that bound
*      abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
*      approximately to the bound that is closest to abs(EMIN).
*      (EMAX is the exponent of the required number RMAX).
*
      LEXP = 1
      EXBITS = 1
10  CONTINUE
      TRY = LEXP*2
      IF( TRY.LE.( -EMIN ) ) THEN
          LEXP = TRY
          EXBITS = EXBITS + 1
          GO TO 10
      END IF
      IF( LEXP.EQ.-EMIN ) THEN
          UEXP = LEXP
      ELSE
          UEXP = TRY
          EXBITS = EXBITS + 1
      END IF
*
*      Now -LEXP is less than or equal to EMIN, and -UEXP is greater
*      than or equal to EMIN. EXBITS is the number of bits needed to
*      store the exponent.
*
      IF( ( UEXP+EMIN ).GT.( -LEXP-EMIN ) ) THEN
          EXPSUM = 2*LEXP
      ELSE
          EXPSUM = 2*UEXP
      END IF
*
*      EXPSUM is the exponent range, approximately equal to
*      EMAX - EMIN + 1 .
*
      EMAX = EXPSUM + EMIN - 1
      NBITS = 1 + EXBITS + P
*
*      NBITS is the total number of bits needed to store a
*      floating-point number.
*
      IF( ( MOD( NBITS, 2 ).EQ.1 ) .AND. ( BETA.EQ.2 ) ) THEN

```

```

*
*      Either there are an odd number of bits used to store a
*      floating-point number, which is unlikely, or some bits are
*      not used in the representation of numbers, which is possible,
*      (e.g. Cray machines) or the mantissa has an implicit bit,
*      (e.g. IEEE machines, Dec Vax machines), which is perhaps the
*      most likely. We have to assume the last alternative.
*      If this is true, then we need to reduce EMAX by one because
*      there must be some way of representing zero in an implicit-bit
*      system. On machines like Cray, we are reducing EMAX by one
*      unnecessarily.
*
      EMAX = EMAX - 1
END IF
*
IF( IEEE ) THEN
*
      Assume we are on an IEEE machine which reserves one exponent
      for infinity and NaN.
*
      EMAX = EMAX - 1
END IF
*
      Now create RMAX, the largest machine number, which should
      be equal to  $(1.0 - \text{BETA}^{**(-P)}) * \text{BETA}^{**\text{EMAX}}$  .
*
      First compute  $1.0 - \text{BETA}^{**(-P)}$ , being careful that the
      result is less than 1.0 .
*
      RECBAS = ONE / BETA
      Z = BETA - ONE
      Y = ZERO
      DO 20 I = 1, P
        Z = Z*RECBAS
        IF( Y.LT.ONE )
          $      OLDY = Y
            Y = DLAMC3( Y, Z )
20 CONTINUE
      IF( Y.GE.ONE )
        $      Y = OLDY
*
      Now multiply by  $\text{BETA}^{**\text{EMAX}}$  to get RMAX.
*
      DO 30 I = 1, EMAX
        Y = DLAMC3( Y*BETA, ZERO )
30 CONTINUE
*
      RMAX = Y
      RETURN
*

```

```

*      End of DLAMC5
*
*      END
*
*

```

— LAPACK dlamc5 —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlamc5 (beta p emin ieee emax rmax)
    (declare (type (double-float) rmax)
              (type (member t nil) ieee)
              (type fixnum emax emin p beta))
    (prog ((oldy 0.0) (recbas 0.0) (y 0.0) (z 0.0) (exbits 0) (expsum 0) (i 0)
           (lexp 0) (nbits 0) (try 0) (uexp 0))
      (declare (type (double-float) oldy recbas y z)
                (type fixnum exbits expsum i lexp nbits try uexp))
      (setf lexp 1)
      (setf exbits 1)
    label10
      (setf try (f2cl-lib:int-mul lexp 2))
      (cond
        ((<= try (f2cl-lib:int-sub emin))
         (setf lexp try)
         (setf exbits (f2cl-lib:int-add exbits 1))
         (go label10)))
      (cond
        ((= lexp (f2cl-lib:int-sub emin))
         (setf uexp lexp))
        (t
         (setf uexp try)
         (setf exbits (f2cl-lib:int-add exbits 1))))
      (cond
        ((> (f2cl-lib:int-add uexp emin)
             (f2cl-lib:int-add (f2cl-lib:int-sub lexp) (f2cl-lib:int-sub emin)))
         (setf expsum (f2cl-lib:int-mul 2 lexp)))
        (t
         (setf expsum (f2cl-lib:int-mul 2 uexp))))
      (setf emax (f2cl-lib:int-sub (f2cl-lib:int-add expsum emin) 1))
      (setf nbits (f2cl-lib:int-add 1 exbits p))
      (cond
        ((and (= (mod nbits 2) 1) (= beta 2))
         (setf emax (f2cl-lib:int-sub emax 1))))
      (cond

```

```

      (ieee
        (setf emax (f2cl-lib:int-sub emax 1))))
    (setf recbas (/ one beta))
    (setf z (- beta one))
    (setf y zero)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i p) nil)
      (tagbody
        (setf z (* z recbas))
        (if (< y one) (setf oldy y))
        (setf y
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 y z)
            (declare (ignore))
            (setf y var-0)
            (setf z var-1)
            ret-val))))
    (if (>= y one) (setf y oldy))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i emax) nil)
      (tagbody
        (setf y
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3 (* y beta) zero)
            (declare (ignore var-0))
            (setf zero var-1)
            ret-val))))
    (setf rmax y)
  end_label
  (return (values beta nil emin ieee emax rmax))))

```

dlamrg LAPACK

— dlamrg.input —

```

)set break resume
)sys rm -f dlamrg.output
)spool dlamrg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlamrg.help —

```
=====
dlamrg examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLAMRG - create a permutation list which will merge the elements of A (which is composed of two independently sorted sets) into a single set which is sorted in ascending order

SYNOPSIS

```
SUBROUTINE DLAMRG( N1, N2, A, DTRD1, DTRD2, INDEX )
```

```
      INTEGER      DTRD1, DTRD2, N1, N2
```

```
      INTEGER      INDEX( * )
```

```
      DOUBLE      PRECISION A( * )
```

Purpose

```
=====
```

DLAMRG will create a permutation list which will merge the elements of A (which is composed of two independently sorted sets) into a single set which is sorted in ascending order.

Arguments

```
=====
```

N1 (input) INTEGER

N2 (input) INTEGER

These arguments contain the respective lengths of the two sorted lists to be merged.

A (input) DOUBLE PRECISION array, dimension (N1+N2)

The first N1 elements of A contain a list of numbers which are sorted in either ascending or descending order. Likewise for the final N2 elements.

DTRD1 (input) INTEGER

DTRD2 (input) INTEGER

These are the strides to be taken through the array A.

Allowable strides are 1 and -1. They indicate whether a subset of A is sorted in ascending (DTRDx = 1) or descending (DTRDx = -1) order.

INDX (output) INTEGER array, dimension (N1+N2)
 On exit this array will contain a permutation such that if $B(I) = A(\text{INDX}(I))$ for $I=1, N1+N2$, then B will be sorted in ascending order.

— dlamrg.f —

```

SUBROUTINE DLAMRG( N1, N2, A, DTRD1, DTRD2, INDX )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    September 30, 1994
*
*    .. Scalar Arguments ..
INTEGER          DTRD1, DTRD2, N1, N2
*
*    ..
*
*    .. Array Arguments ..
INTEGER          INDX( * )
DOUBLE PRECISION A( * )
*
*    ..
*
*  =====
*
*    .. Local Scalars ..
INTEGER          I, IND1, IND2, N1SV, N2SV
*
*    ..
*
*    .. Executable Statements ..
*
    N1SV = N1
    N2SV = N2
    IF( DTRD1.GT.0 ) THEN
        IND1 = 1
    ELSE
        IND1 = N1
    END IF
    IF( DTRD2.GT.0 ) THEN
        IND2 = 1 + N1
    ELSE
        IND2 = N1 + N2
    END IF
    I = 1

```



```

*      while ( (N1SV > 0) & (N2SV > 0) )
10  CONTINUE
    IF( N1SV.GT.0 .AND. N2SV.GT.0 ) THEN
      IF( A( IND1 ).LE.A( IND2 ) ) THEN
        INDX( I ) = IND1
        I = I + 1
        IND1 = IND1 + DTRD1
        N1SV = N1SV - 1
      ELSE
        INDX( I ) = IND2
        I = I + 1
        IND2 = IND2 + DTRD2
        N2SV = N2SV - 1
      END IF
      GO TO 10
    END IF
  end while
*
  IF( N1SV.EQ.0 ) THEN
    DO 20 N1SV = 1, N2SV
      INDX( I ) = IND2
      I = I + 1
      IND2 = IND2 + DTRD2
20  CONTINUE
  ELSE
*
    N2SV .EQ. 0
    DO 30 N2SV = 1, N1SV
      INDX( I ) = IND1
      I = I + 1
      IND1 = IND1 + DTRD1
30  CONTINUE
  END IF
*
  RETURN
*
*      End of DLAMRG
*
  END

```

— LAPACK dlamrg —

```

(defun dlamrg (n1 n2 a dtrd1 dtrd2 indx)
  (declare (type (simple-array fixnum (*)) indx)
            (type (simple-array double-float (*)) a)
            (type fixnum dtrd2 dtrd1 n2 n1))
  (f2cl-lib:with-multi-array-data
    ((a double-float a-%data% a-%offset%)

```

```

      (indx fixnum indx-%data% indx-%offset%))
(prog ((i 0) (ind1 0) (ind2 0) (n1sv 0) (n2sv 0))
  (declare (type fixnum n2sv n1sv ind2 ind1 i))
  (setf n1sv n1)
  (setf n2sv n2)
  (cond
    ((> dtrd1 0)
     (setf ind1 1))
    (t
     (setf ind1 n1)))
  (cond
    ((> dtrd2 0)
     (setf ind2 (f2cl-lib:int-add 1 n1)))
    (t
     (setf ind2 (f2cl-lib:int-add n1 n2))))
  (setf i 1)
label10
  (cond
    ((and (> n1sv 0) (> n2sv 0))
     (cond
      ((<= (f2cl-lib:fref a (ind1) ((1 *)))
            (f2cl-lib:fref a (ind2) ((1 *))))
       (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind1)
       (setf i (f2cl-lib:int-add i 1))
       (setf ind1 (f2cl-lib:int-add ind1 dtrd1))
       (setf n1sv (f2cl-lib:int-sub n1sv 1)))
      (t
       (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind2)
       (setf i (f2cl-lib:int-add i 1))
       (setf ind2 (f2cl-lib:int-add ind2 dtrd2))
       (setf n2sv (f2cl-lib:int-sub n2sv 1))))
     (go label10)))
  (cond
    ((= n1sv 0)
     (f2cl-lib:fdo (n1sv 1 (f2cl-lib:int-add n1sv 1))
                   ((> n1sv n2sv) nil)
     (tagbody
      (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind2)
      (setf i (f2cl-lib:int-add i 1))
      (setf ind2 (f2cl-lib:int-add ind2 dtrd2))))))
    (t
     (f2cl-lib:fdo (n2sv 1 (f2cl-lib:int-add n2sv 1))
                   ((> n2sv n1sv) nil)
     (tagbody
      (setf (f2cl-lib:fref indx-%data% (i) ((1 *)) indx-%offset%) ind1)
      (setf i (f2cl-lib:int-add i 1))
      (setf ind1 (f2cl-lib:int-add ind1 dtrd1))))))
end_label
  (return (values nil nil nil nil nil nil)))

```

dlange LAPACK

— dlange.input —

```
)set break resume
)sys rm -f dlange.output
)spool dlange.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlange.help —

```
=====
dlange examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLANGE - the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A

SYNOPSIS

```
DOUBLE PRECISION FUNCTION DLANGE( NORM, M, N, A, LDA, WORK )
```

```
CHARACTER    NORM
```

```
INTEGER      LDA, M, N
```

```
DOUBLE       PRECISION A( LDA, * ), WORK( * )
```

PURPOSE

DLANGE returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A.

DESCRIPTION

DLANGE returns the value

```
DLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
          (
            ( norm1(A),          NORM = '1', 'O' or 'o'
              (
                ( normI(A),      NORM = 'I' or 'i'
                  (
                    ( normF(A),   NORM = 'F', 'f', 'E' or 'e'
```

where `norm1` denotes the one norm of a matrix (maximum column sum), `normI` denotes the infinity norm of a matrix (maximum row sum) and `normF` denotes the Frobenius norm of a matrix (square root of sum of squares). Note that `max(abs(A(i,j)))` is not a consistent matrix norm.

ARGUMENTS

NORM (input) CHARACTER*1
Specifies the value to be returned in DLANGE as described above.

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$. When $M = 0$, DLANGE is set to zero.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$. When $N = 0$, DLANGE is set to zero.

A (input) DOUBLE PRECISION array, dimension (LDA,N)
The m by n matrix A.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(M,1)$.

WORK (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK)), where $LWORK \geq M$ when $NORM = 'I'$; otherwise, WORK is not referenced.

— dlange.f —

DOUBLE PRECISION FUNCTION DLANGE(NORM, M, N, A, LDA, WORK)

```

*
* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   October 31, 1992
*
*   .. Scalar Arguments ..
*   CHARACTER          NORM
*   INTEGER            LDA, M, N
*
*   ..
*   .. Array Arguments ..
*   DOUBLE PRECISION   A( LDA, * ), WORK( * )
*
*   ..
*
* =====
*
*   .. Parameters ..
*   DOUBLE PRECISION   ONE, ZERO
*   PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*   ..
*   .. Local Scalars ..
*   INTEGER            I, J
*   DOUBLE PRECISION   SCALE, SUM, VALUE
*
*   ..
*   .. External Subroutines ..
*   EXTERNAL           DLASSQ
*
*   ..
*   .. External Functions ..
*   LOGICAL            LSAME
*   EXTERNAL           LSAME
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC          ABS, MAX, MIN, SQRT
*
*   ..
*   .. Executable Statements ..
*
*   IF( MIN( M, N ).EQ.0 ) THEN
*       VALUE = ZERO
*   ELSE IF( LSAME( NORM, 'M' ) ) THEN
*
*       Find max(abs(A(i,j))).
*
*       VALUE = ZERO
*       DO 20 J = 1, N
*           DO 10 I = 1, M
*               VALUE = MAX( VALUE, ABS( A( I, J ) ) )
*           10 CONTINUE
*       20 CONTINUE
*   ELSE IF( ( LSAME( NORM, 'O' ) ) .OR. ( NORM.EQ.'1' ) ) THEN
*

```

```

*      Find norm1(A).
*
      VALUE = ZERO
      DO 40 J = 1, N
        SUM = ZERO
        DO 30 I = 1, M
          SUM = SUM + ABS( A( I, J ) )
30      CONTINUE
        VALUE = MAX( VALUE, SUM )
40      CONTINUE
      ELSE IF( LSAME( NORM, 'I' ) ) THEN
*
*      Find normI(A).
*
      DO 50 I = 1, M
        WORK( I ) = ZERO
50      CONTINUE
      DO 70 J = 1, N
        DO 60 I = 1, M
          WORK( I ) = WORK( I ) + ABS( A( I, J ) )
60      CONTINUE
70      CONTINUE
      VALUE = ZERO
      DO 80 I = 1, M
        VALUE = MAX( VALUE, WORK( I ) )
80      CONTINUE
      ELSE IF( ( LSAME( NORM, 'F' ) ) .OR. ( LSAME( NORM, 'E' ) ) ) THEN
*
*      Find normF(A).
*
      SCALE = ZERO
      SUM = ONE
      DO 90 J = 1, N
        CALL DLAASSQ( M, A( 1, J ), 1, SCALE, SUM )
90      CONTINUE
      VALUE = SCALE*SQRT( SUM )
      END IF
*
      DLANGE = VALUE
      RETURN
*
*      End of DLANGE
*
      END

```



```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%
      zero)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
          (+
            (f2cl-lib:fref work-%data%
              (i)
              ((1 *))
              work-%offset%)
            (abs
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))))))
        (setf value zero)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
          (tagbody
            (setf value
              (max value
                (f2cl-lib:fref work-%data%
                  (i)
                  ((1 *))
                  work-%offset%))))))
        ((or (char-equal norm #\F) (char-equal norm #\E))
          (setf scale zero)
          (setf sum one)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            (> j n) nil)
            (tagbody
              (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                (dlassq m
                  (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *)))
                  1 scale sum)
                (declare (ignore var-0 var-1 var-2))
                (setf scale var-3)
                (setf sum var-4))))
              (setf value (* scale (f2cl-lib:fsqrt sum))))
          (setf dlange value)
          end_label
          (return (values dlange nil nil nil nil nil nil))))))

```


dlanhs LAPACK

— dlanhs.input —

```
)set break resume
)sys rm -f dlanhs.output
)spool dlanhs.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlanhs.help —

```
=====
dlanhs examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLANHS - the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A

SYNOPSIS

DOUBLE PRECISION FUNCTION DLANHS(NORM, N, A, LDA, WORK)

CHARACTER NORM

INTEGER LDA, N

DOUBLE PRECISION A(LDA, *), WORK(*)

PURPOSE

DLANHS returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A.

DESCRIPTION

DLANHS returns the value

```
DLANHS = ( max(abs(A(i,j))), NORM = 'M' or 'm'
          (
            ( norm1(A),          NORM = '1', 'O' or 'o'
              (
                ( normI(A),      NORM = 'I' or 'i'
                  (
                    ( normF(A),   NORM = 'F', 'f', 'E' or 'e'
```

where `norm1` denotes the one norm of a matrix (maximum column sum), `normI` denotes the infinity norm of a matrix (maximum row sum) and `normF` denotes the Frobenius norm of a matrix (square root of sum of squares). Note that `max(abs(A(i,j)))` is not a consistent matrix norm.

ARGUMENTS

NORM (input) CHARACTER*1
Specifies the value to be returned in DLANHS as described above.

N (input) INTEGER
The order of the matrix A. $N \geq 0$. When $N = 0$, DLANHS is set to zero.

A (input) DOUBLE PRECISION array, dimension (LDA,N)
The n by n upper Hessenberg matrix A; the part of A below the first sub-diagonal is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(N, 1)$.

WORK (workspace) DOUBLE PRECISION array, dimension (MAX(1,LWORK)), where $LWORK \geq N$ when $NORM = 'I'$; otherwise, WORK is not referenced.

— dlanhs.f —

DOUBLE PRECISION FUNCTION DLANHS(NORM, N, A, LDA, WORK)

```
*
* -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
```

```
* Courant Institute, Argonne National Lab, and Rice University
* October 31, 1992
*
```

```
* .. Scalar Arguments ..
* CHARACTER          NORM
* INTEGER            LDA, N
* ..
* .. Array Arguments ..
* DOUBLE PRECISION   A( LDA, * ), WORK( * )
* ..
*
```

```
* =====
```

```
* .. Parameters ..
* DOUBLE PRECISION   ONE, ZERO
* PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
```

```
* .. Local Scalars ..
* INTEGER            I, J
* DOUBLE PRECISION   SCALE, SUM, VALUE
*
```

```
* .. External Subroutines ..
* EXTERNAL           DLASSQ
*
```

```
* .. External Functions ..
* LOGICAL            LSAME
* EXTERNAL           LSAME
*
```

```
* .. Intrinsic Functions ..
* INTRINSIC          ABS, MAX, MIN, SQRT
*
```

```
* .. Executable Statements ..
*
```

```
IF( N.EQ.0 ) THEN
  VALUE = ZERO
ELSE IF( LSAME( NORM, 'M' ) ) THEN
```

```
* Find max(abs(A(i,j))).
*
```

```
VALUE = ZERO
DO 20 J = 1, N
  DO 10 I = 1, MIN( N, J+1 )
    VALUE = MAX( VALUE, ABS( A( I, J ) ) )
```

```
10 CONTINUE
```

```
20 CONTINUE
```

```
ELSE IF( ( LSAME( NORM, 'O' ) ) .OR. ( NORM.EQ.'1' ) ) THEN
```

```
* Find norm1(A).
*
```

```
VALUE = ZERO
```

```

      DO 40 J = 1, N
        SUM = ZERO
        DO 30 I = 1, MIN( N, J+1 )
          SUM = SUM + ABS( A( I, J ) )
30      CONTINUE
        VALUE = MAX( VALUE, SUM )
40      CONTINUE
      ELSE IF( LSAME( NORM, 'I' ) ) THEN
*
*      Find normI(A).
*
      DO 50 I = 1, N
        WORK( I ) = ZERO
50      CONTINUE
      DO 70 J = 1, N
        DO 60 I = 1, MIN( N, J+1 )
          WORK( I ) = WORK( I ) + ABS( A( I, J ) )
60      CONTINUE
70      CONTINUE
        VALUE = ZERO
      DO 80 I = 1, N
        VALUE = MAX( VALUE, WORK( I ) )
80      CONTINUE
      ELSE IF( ( LSAME( NORM, 'F' ) ) .OR. ( LSAME( NORM, 'E' ) ) ) THEN
*
*      Find normF(A).
*
      SCALE = ZERO
      SUM = ONE
      DO 90 J = 1, N
        CALL DLASSQ( MIN( N, J+1 ), A( 1, J ), 1, SCALE, SUM )
90      CONTINUE
        VALUE = SCALE*SQRT( SUM )
      END IF
*
      DLANHS = VALUE
      RETURN
*
*      End of DLANHS
*
      END

```

— LAPACK dlanhs —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)

```



```

                                (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%))))))
    (setf value (max value sum))))))
((char-equal norm #\I)
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
               (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%
                        zero)))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                  (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      (> i
                        (min (the fixnum n)
                             (the fixnum
                               (f2cl-lib:int-add j 1))))
                      nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%
                                (+
                                  (f2cl-lib:fref work-%data%
                                                    (i)
                                                    ((1 *))
                                                    work-%offset%)
                                  (abs
                                    (f2cl-lib:fref a-%data%
                                                    (i j)
                                                    ((1 lda) (1 *))
                                                    a-%offset%)))))))
            (setf value zero)
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                          (> i n) nil)
              (tagbody
                (setf value
                      (max value
                        (f2cl-lib:fref work-%data%
                                      (i)
                                      ((1 *))
                                      work-%offset%))))))
              ((or (char-equal norm #\F) (char-equal norm #\E))
                (setf scale zero)
                (setf sum one)
                (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                              (> j n) nil)
                  (tagbody
                    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
                      (dlassq

```

```

      (min (the fixnum n)
            (the fixnum (f2cl-lib:int-add j 1)))
      (f2cl-lib:array-slice a double-float (1 j) ((1 lda) (1 *)))
      1 scale sum)
      (declare (ignore var-0 var-1 var-2))
      (setf scale var-3)
      (setf sum var-4)))
      (setf value (* scale (f2cl-lib:fsqrt sum))))
      (setf dlanhs value)
end_label
      (return (values dlanhs nil nil nil nil nil))))

```

dlanst LAPACK

— dlanst.input —

```

)set break resume
)sys rm -f dlanst.output
)spool dlanst.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlanst.help —

```

=====
dlanst examples
=====

=====
Man Page Details
=====

```

NAME

DLANST - the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A

SYNOPSIS

DOUBLE PRECISION FUNCTION DLANST(NORM, N, D, E)

CHARACTER NORM

INTEGER N

DOUBLE PRECISION D(*), E(*)

PURPOSE

DLANST returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A.

DESCRIPTION

DLANST returns the value

```
DLANST = ( max(abs(A(i,j))), NORM = 'M' or 'm'
          (
            ( norm1(A),          NORM = '1', 'O' or 'o'
              (
                ( normI(A),      NORM = 'I' or 'i'
                  (
                    ( normF(A),   NORM = 'F', 'f', 'E' or 'e'
```

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

ARGUMENTS

NORM (input) CHARACTER*1
 Specifies the value to be returned in DLANST as described above.

N (input) INTEGER
 The order of the matrix A. N >= 0. When N = 0, DLANST is set to zero.

D (input) DOUBLE PRECISION array, dimension (N)
 The diagonal elements of A.

E (input) DOUBLE PRECISION array, dimension (N-1)
 The (n-1) sub-diagonal or super-diagonal elements of A.

— dlanst.f —

```

      DOUBLE PRECISION FUNCTION DLANST( NORM, N, D, E )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
      CHARACTER          NORM
      INTEGER            N
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION   D( * ), E( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION   ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
      INTEGER            I
      DOUBLE PRECISION   ANORM, SCALE, SUM
*
*  ..
*
*  .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*
*  ..
*
*  .. External Subroutines ..
      EXTERNAL           DCLASSQ
*
*  ..
*
*  .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, SQRT
*
*  ..
*
*  .. Executable Statements ..
*
      IF( N.LE.0 ) THEN
         ANORM = ZERO
      ELSE IF( LSAME( NORM, 'M' ) ) THEN
*
*         Find max(abs(A(i,j))).
*
*
         ANORM = ABS( D( N ) )
         DO 10 I = 1, N - 1
            ANORM = MAX( ANORM, ABS( D( I ) ) )
            ANORM = MAX( ANORM, ABS( E( I ) ) )

```

```

10    CONTINUE
      ELSE IF( LSAME( NORM, 'O' ) .OR. NORM.EQ.'1' .OR.
$         LSAME( NORM, 'I' ) ) THEN
*
*       Find norm1(A).
*
      IF( N.EQ.1 ) THEN
        ANORM = ABS( D( 1 ) )
      ELSE
        ANORM = MAX( ABS( D( 1 ) )+ABS( E( 1 ) ),
$           ABS( E( N-1 ) )+ABS( D( N ) ) )
        DO 20 I = 2, N - 1
          ANORM = MAX( ANORM, ABS( D( I ) )+ABS( E( I ) )+
$             ABS( E( I-1 ) ) )
20    CONTINUE
      END IF
      ELSE IF( ( LSAME( NORM, 'F' ) ) .OR. ( LSAME( NORM, 'E' ) ) ) THEN
*
*       Find normF(A).
*
      SCALE = ZERO
      SUM = ONE
      IF( N.GT.1 ) THEN
        CALL DCLASSQ( N-1, E, 1, SCALE, SUM )
        SUM = 2*SUM
      END IF
      CALL DCLASSQ( N, D, 1, SCALE, SUM )
      ANORM = SCALE*SQRT( SUM )
      END IF
*
      DLANST = ANORM
      RETURN
*
*     End of DLANST
*
      END

```

— LAPACK dlanst —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dlanst (norm n d e)
    (declare (type (simple-array double-float (*)) e d)
              (type fixnum n)
              (type character norm))

```

```

(f2cl-lib:with-multi-array-data
  ((norm character norm-%data% norm-%offset%)
   (d double-float d-%data% d-%offset%)
   (e double-float e-%data% e-%offset%))
  (prog ((anorm 0.0) (scale 0.0) (sum 0.0) (i 0) (dlanst 0.0))
    (declare (type fixnum i)
              (type (double-float) anorm scale sum dlanst))
    (cond
      ((<= n 0)
       (setf anorm zero))
      ((char-equal norm #\M)
       (setf anorm (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
       (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
                     (> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
       (tagbody
        (setf anorm
              (max anorm
                   (abs
                    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
        (setf anorm
              (max anorm
                   (abs
                    (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))))))
      ((or (char-equal norm #\0) (f2cl-lib:fstring-= norm "1")
           (char-equal norm #\I))
       (cond
         ((= n 1)
          (setf anorm
                (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))))
         (t
          (setf anorm
                (max
                 (+ (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
                    (abs
                     (f2cl-lib:fref e-%data% (1) ((1 *)) e-%offset%)))
                 (+
                  (abs
                   (f2cl-lib:fref e-%data%
                                 ((f2cl-lib:int-sub n 1))
                                 ((1 *))
                                 e-%offset%))
                  (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))))))
          (f2cl-lib:fd0 (i 2 (f2cl-lib:int-add i 1))
                        (> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                        nil)
          (tagbody
           (setf anorm
                 (max anorm
                      (+
                       (abs

```

```

(f2cl-lib:fref d-%data%
  (i)
  ((1 *))
  d-%offset%))
(abs
  (f2cl-lib:fref e-%data%
    (i)
    ((1 *))
    e-%offset%))
(abs
  (f2cl-lib:fref e-%data%
    ((f2cl-lib:int-sub i 1))
    ((1 *))
    e-%offset%)))))))))
((or (char-equal norm #\F) (char-equal norm #\E))
  (setf scale zero)
  (setf sum one)
  (cond
    (> n 1)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlassq (f2cl-lib:int-sub n 1) e 1 scale sum)
      (declare (ignore var-0 var-1 var-2))
      (setf scale var-3)
      (setf sum var-4))
    (setf sum (* 2 sum))))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
    (dlassq n d 1 scale sum)
    (declare (ignore var-0 var-1 var-2))
    (setf scale var-3)
    (setf sum var-4))
  (setf anorm (* scale (f2cl-lib:fsqrt sum))))
  (setf dlanst anorm)
end_label
  (return (values dlanst nil nil nil nil))))))

```

dlanv2 LAPACK

— dlanv2.input —

```

)set break resume
)sys rm -f dlanv2.output
)spool dlanv2.output
)set message test on
)set message auto off

```

```
)clear all
```

```
)spool
```

```
)lisp (bye)
```

— dlanv2.help —

```
=====
dlanv2 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLANV2 - the Schur factorization of a real 2-by-2 nonsymmetric matrix
in standard form

SYNOPSIS

```
SUBROUTINE DLANV2( A, B, C, D, RT1R, RT1I, RT2R, RT2I, CS, SN )
```

```
DOUBLE          PRECISION A, B, C, CS, D, RT1I, RT1R, RT2I, RT2R, SN
```

Purpose

```
=====
```

DLANV2 computes the Schur factorization of a real 2-by-2 nonsymmetric
matrix in standard form:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} CS & -SN \\ SN & CS \end{bmatrix} \begin{bmatrix} AA & BB \\ CC & DD \end{bmatrix} \begin{bmatrix} CS & SN \\ -SN & CS \end{bmatrix}$$

where either

- 1) $CC = 0$ so that AA and DD are real eigenvalues of the matrix, or
- 2) $AA = DD$ and $BB*CC < 0$, so that $AA + \text{or} - \sqrt{BB*CC}$ are complex conjugate eigenvalues.

Arguments

```
=====
```

A (input/output) DOUBLE PRECISION

B (input/output) DOUBLE PRECISION

C (input/output) DOUBLE PRECISION

D (input/output) DOUBLE PRECISION

On entry, the elements of the input matrix.

On exit, they are overwritten by the elements of the

standardised Schur form.

RT1R (output) DOUBLE PRECISION
 RT1I (output) DOUBLE PRECISION
 RT2R (output) DOUBLE PRECISION
 RT2I (output) DOUBLE PRECISION
 The real and imaginary parts of the eigenvalues. If the
 eigenvalues are a complex conjugate pair, RT1I > 0.

CS (output) DOUBLE PRECISION
 SN (output) DOUBLE PRECISION
 Parameters of the rotation matrix.

Further Details

=====

Modified by V. Sima, Research Institute for Informatics, Bucharest,
 Romania, to reduce the risk of cancellation errors,
 when computing real eigenvalues, and to ensure, if possible, that
 $\text{abs}(\text{RT1R}) \geq \text{abs}(\text{RT2R})$.

— dlanv2.f —

```

SUBROUTINE DLANV2( A, B, C, D, RT1R, RT1I, RT2R, RT2I, CS, SN )
*
*  -- LAPACK driver routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    June 30, 1999
*
*    .. Scalar Arguments ..
*    DOUBLE PRECISION  A, B, C, CS, D, RT1I, RT1R, RT2I, RT2R, SN
*    ..
*
*    =====
*
*    .. Parameters ..
*    DOUBLE PRECISION  ZERO, HALF, ONE
*    PARAMETER          ( ZERO = 0.0D+0, HALF = 0.5D+0, ONE = 1.0D+0 )
*    DOUBLE PRECISION  MULTPL
*    PARAMETER          ( MULTPL = 4.0D+0 )
*
*    ..
*    .. Local Scalars ..
*    DOUBLE PRECISION  AA, BB, BCMAX, BCMIS, CC, CS1, DD, EPS, P, SAB,
*    $                 SAC, SCALE, SIGMA, SN1, TAU, TEMP, Z
*    ..

```

```

*      .. External Functions ..
      DOUBLE PRECISION  DLAMCH, DLAPY2
      EXTERNAL          DLAMCH, DLAPY2
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, MIN, SIGN, SQRT
*
*      ..
*      .. Executable Statements ..
*
      EPS = DLAMCH( 'P' )
      IF( C.EQ.ZERO ) THEN
        CS = ONE
        SN = ZERO
        GO TO 10
*
      ELSE IF( B.EQ.ZERO ) THEN
*
*      Swap rows and columns
*
*
        CS = ZERO
        SN = ONE
        TEMP = D
        D = A
        A = TEMP
        B = -C
        C = ZERO
        GO TO 10
      ELSE IF( ( A-D ).EQ.ZERO .AND. SIGN( ONE, B ).NE.SIGN( ONE, C ) )
$      THEN
        CS = ONE
        SN = ZERO
        GO TO 10
      ELSE
*
        TEMP = A - D
        P = HALF*TEMP
        BCMAX = MAX( ABS( B ), ABS( C ) )
        BCMIS = MIN( ABS( B ), ABS( C ) )*SIGN( ONE, B )*SIGN( ONE, C )
        SCALE = MAX( ABS( P ), BCMAX )
        Z = ( P / SCALE )*P + ( BCMAX / SCALE )*BCMIS
*
*      If Z is of the order of the machine accuracy, postpone the
*      decision on the nature of eigenvalues
*
      IF( Z.GE.MULTPL*EPS ) THEN
*
*      Real eigenvalues. Compute A and D.
*
*
        Z = P + SIGN( SQRT( SCALE )*SQRT( Z ), P )
        A = D + Z

```

```

D = D - ( BCMAX / Z )*BCMIS
*
*   Compute B and the rotation matrix
*
TAU = DLAPY2( C, Z )
CS = Z / TAU
SN = C / TAU
B = B - C
C = ZERO
ELSE
*
*   Complex eigenvalues, or real (almost) equal eigenvalues.
*   Make diagonal elements equal.
*
SIGMA = B + C
TAU = DLAPY2( SIGMA, TEMP )
CS = SQRT( HALF*( ONE+ABS( SIGMA ) / TAU ) )
SN = -( P / ( TAU*CS ) )*SIGN( ONE, SIGMA )
*
*   Compute [ AA  BB ] = [ A  B ] [ CS -SN ]
*           [ CC  DD ]   [ C  D ] [ SN  CS ]
*
AA = A*CS + B*SN
BB = -A*SN + B*CS
CC = C*CS + D*SN
DD = -C*SN + D*CS
*
*   Compute [ A  B ] = [ CS  SN ] [ AA  BB ]
*           [ C  D ]   [-SN  CS ] [ CC  DD ]
*
A = AA*CS + CC*SN
B = BB*CS + DD*SN
C = -AA*SN + CC*CS
D = -BB*SN + DD*CS
*
TEMP = HALF*( A+D )
A = TEMP
D = TEMP
*
IF( C.NE.ZERO ) THEN
  IF( B.NE.ZERO ) THEN
    IF( SIGN( ONE, B ).EQ.SIGN( ONE, C ) ) THEN
*
*   Real eigenvalues: reduce to upper triangular form
*
SAB = SQRT( ABS( B ) )
SAC = SQRT( ABS( C ) )
P = SIGN( SAB*SAC, C )
TAU = ONE / SQRT( ABS( B+C ) )
A = TEMP + P

```



```

        D = TEMP - P
        B = B - C
        C = ZERO
        CS1 = SAB*TAU
        SN1 = SAC*TAU
        TEMP = CS*CS1 - SN*SN1
        SN = CS*SN1 + SN*CS1
        CS = TEMP
    END IF
ELSE
    B = -C
    C = ZERO
    TEMP = CS
    CS = -SN
    SN = TEMP
END IF
END IF
END IF
*
    END IF
*
10 CONTINUE
*
    Store eigenvalues in (RT1R,RT1I) and (RT2R,RT2I).
*
    RT1R = A
    RT2R = D
    IF( C.EQ.ZERO ) THEN
        RT1I = ZERO
        RT2I = ZERO
    ELSE
        RT1I = SQRT( ABS( B ) ) * SQRT( ABS( C ) )
        RT2I = -RT1I
    END IF
    RETURN
*
*   End of DLANV2
*
END

```

— LAPACK dlanv2 —

```

(let* ((zero 0.0) (half 0.5) (one 1.0) (multpl 4.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 0.5 0.5) half)
            (type (double-float 1.0 1.0) one)

```

```

(type (double-float 4.0 4.0) multpl))
(defun dlanv2 (a b c d rt1r rt1i rt2r rt2i cs sn)
  (declare (type (double-float) sn cs rt2i rt2r rt1i rt1r d c b a))
  (prog ((aa 0.0) (bb 0.0) (bcmax 0.0) (bcmis 0.0) (cc 0.0) (cs1 0.0)
        (dd 0.0) (eps 0.0) (p 0.0) (sab 0.0) (sac 0.0) (scale 0.0)
        (sigma 0.0) (sn1 0.0) (tau 0.0) (temp 0.0) (z 0.0))
    (declare (type (double-float) aa bb bcmax bcmis cc cs1 dd eps p sab sac
                  scale sigma sn1 tau temp z))
    (setf eps (dlamch "P"))
    (cond
      ((= c zero)
       (setf cs one)
       (setf sn zero)
       (go label10))
      ((= b zero)
       (setf cs zero)
       (setf sn one)
       (setf temp d)
       (setf d a)
       (setf a temp)
       (setf b (- c))
       (setf c zero)
       (go label10))
      (and (= (+ a (- d)) zero)
           (/= (f2cl-lib:sign one b) (f2cl-lib:sign one c)))
       (setf cs one)
       (setf sn zero)
       (go label10))
      (t
       (setf temp (- a d))
       (setf p (* half temp))
       (setf bcmax (max (abs b) (abs c)))
       (setf bcmis
        (* (min (abs b) (abs c))
           (f2cl-lib:sign one b)
           (f2cl-lib:sign one c)))
       (setf scale (max (abs p) bcmax))
       (setf z (+ (* (/ p scale) p) (* (/ bcmax scale) bcmis)))
       (cond
        ((>= z (* multpl eps))
         (setf z
          (+ p
             (f2cl-lib:sign
              (* (f2cl-lib:fsqrt scale) (f2cl-lib:fsqrt z))
              p)))
         (setf a (+ d z))
         (setf d (- d (* (/ bcmax z) bcmis)))
         (setf tau (dlapy2 c z))
         (setf cs (/ z tau))
         (setf sn (/ c tau))

```

```

      (setf b (- b c))
      (setf c zero))
    (t
      (setf sigma (+ b c))
      (setf tau (dlapy2 sigma temp))
      (setf cs (f2cl-lib:fsqrt (* half (+ one (/ (abs sigma) tau)))))
      (setf sn (* (- (/ p (* tau cs))) (f2cl-lib:sign one sigma)))
      (setf aa (+ (* a cs) (* b sn)))
      (setf bb (+ (* (- a) sn) (* b cs)))
      (setf cc (+ (* c cs) (* d sn)))
      (setf dd (+ (* (- c) sn) (* d cs)))
      (setf a (+ (* aa cs) (* cc sn)))
      (setf b (+ (* bb cs) (* dd sn)))
      (setf c (+ (* (- aa) sn) (* cc cs)))
      (setf d (+ (* (- bb) sn) (* dd cs)))
      (setf temp (* half (+ a d)))
      (setf a temp)
      (setf d temp)
      (cond
        ((/= c zero)
          (cond
            ((/= b zero)
              (cond
                ((= (f2cl-lib:sign one b) (f2cl-lib:sign one c))
                  (setf sab (f2cl-lib:fsqrt (abs b)))
                  (setf sac (f2cl-lib:fsqrt (abs c)))
                  (setf p (f2cl-lib:sign (* sab sac) c))
                  (setf tau (/ one (f2cl-lib:fsqrt (abs (+ b c)))))
                  (setf a (+ temp p))
                  (setf d (- temp p))
                  (setf b (- b c))
                  (setf c zero)
                  (setf cs1 (* sab tau))
                  (setf sn1 (* sac tau))
                  (setf temp (- (* cs cs1) (* sn sn1)))
                  (setf sn (+ (* cs sn1) (* sn cs1)))
                  (setf cs temp))))
                (t
                  (setf b (- c))
                  (setf c zero)
                  (setf temp cs)
                  (setf cs (- sn))
                  (setf sn temp))))))))))
    label10
      (setf rt1r a)
      (setf rt2r d)
      (cond
        ((= c zero)
          (setf rt1i zero)
          (setf rt2i zero))

```

```

      (t
        (setf rt1i (* (f2cl-lib:fsqrt (abs b)) (f2cl-lib:fsqrt (abs c))))
        (setf rt2i (- rt1i))))
    end_label
    (return (values a b c d rt1r rt1i rt2r rt2i cs sn))))

```

dlapy2 LAPACK

— dlapy2.input —

```

)set break resume
)sys rm -f dlapy2.output
)spool dlapy2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlapy2.help —

```

=====
dlapy2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLAPY2 - $\sqrt{x^2+y^2}$, taking care not to cause unnecessary overflow

SYNOPSIS

DOUBLE PRECISION FUNCTION DLAPY2(X, Y)

DOUBLE PRECISION X, Y

Purpose

=====

DLAPY2 returns $\sqrt{x^2+y^2}$, taking care not to cause unnecessary overflow.

Arguments

=====

X (input) DOUBLE PRECISION
 Y (input) DOUBLE PRECISION
 X and Y specify the values x and y.

— dlapy2.f —

```

      DOUBLE PRECISION FUNCTION DLAPY2( X, Y )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1992
*
*    .. Scalar Arguments ..
*    DOUBLE PRECISION   X, Y
*    ..
*
*  =====
*
*    .. Parameters ..
*    DOUBLE PRECISION   ZERO
*    PARAMETER          ( ZERO = 0.0D0 )
*    DOUBLE PRECISION   ONE
*    PARAMETER          ( ONE = 1.0D0 )
*    ..
*    .. Local Scalars ..
*    DOUBLE PRECISION   W, XABS, YABS, Z
*    ..
*    .. Intrinsic Functions ..
*    INTRINSIC          ABS, MAX, MIN, SQRT
*    ..
*    .. Executable Statements ..
*
*    XABS = ABS( X )
*    YABS = ABS( Y )
*    W = MAX( XABS, YABS )
*    Z = MIN( XABS, YABS )
*    IF( Z.EQ.ZERO ) THEN
*       DLAPY2 = W
*    ELSE

```

```

        DLAPY2 = W*SQRT( ONE+( Z / W )**2 )
    END IF
    RETURN
*
*      End of DLAPY2
*
    END

```

— LAPACK dlapy2 —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
           (type (double-float 1.0 1.0) one))
  (defun dlapy2 (x y)
    (declare (type (double-float) y x))
    (prog ((w 0.0) (xabs 0.0) (yabs 0.0) (z 0.0) (dlapy2 0.0))
      (declare (type (double-float) w xabs yabs z dlapy2))
      (setf xabs (abs x))
      (setf yabs (abs y))
      (setf w (max xabs yabs))
      (setf z (min xabs yabs))
      (cond
        ((= z zero)
         (setf dlapy2 w))
        (t
         (setf dlapy2 (* w (f2cl-lib:fsqrt (+ one (expt (/ z w) 2))))))
      )
      (return (values dlapy2 nil nil))))

```

dlapy3 LAPACK

— dlapy3.input —

```

)set break resume
)sys rm -f dlapy3.output
)spool dlapy3.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— dlapy3.help —

=====

dlapy3 examples

=====

=====

Man Page Details

=====

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

DOUBLE PRECISION FUNCTION DLAPY3(X, Y, Z)

.. Scalar Arguments ..
 DOUBLE PRECISION X, Y, Z
 ..

Purpose:
 =====

DLAPY3 returns $\sqrt{x^2+y^2+z^2}$, taking care not to cause unnecessary overflow.

Arguments:
 =====

[in] X
 X is DOUBLE PRECISION

[in] Y
 Y is DOUBLE PRECISION

[in] Z
 Z is DOUBLE PRECISION
 X, Y and Z specify the values x, y and z.

Authors:
 =====
 Univ. of Tennessee

Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— dlapy3.f —

```
* =====
*      DOUBLE PRECISION FUNCTION DLAPY3( X, Y, Z )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      DOUBLE PRECISION    X, Y, Z
*      ..
*
* =====
*
*      .. Parameters ..
*      DOUBLE PRECISION    ZERO
*      PARAMETER            ( ZERO = 0.0D0 )
*      ..
*      .. Local Scalars ..
*      DOUBLE PRECISION    W, XABS, YABS, ZABS
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC            ABS, MAX, SQRT
*      ..
*      .. Executable Statements ..
*
*      XABS = ABS( X )
*      YABS = ABS( Y )
*      ZABS = ABS( Z )
*      W = MAX( XABS, YABS, ZABS )
*      IF( W.EQ.ZERO ) THEN
*      W can be zero for max(0,nan,0)
*      adding all three entries together will make sure
*      NaN will not disappear.
*          DLAPY3 = XABS + YABS + ZABS
*      ELSE
*          DLAPY3 = W*SQRT( ( XABS / W )**2+( YABS / W )**2+
*      $              ( ZABS / W )**2 )
```



```

        END IF
        RETURN
*
*      End of DLAPY3
*
      END

```

— LAPACK dlapy3 —

```

(let* ((zero 0.0d0))
  (declare (type (double-float 0.0d0 0.0d0) zero) (ignorable zero))
  (defun dlapy3 (x y z) (declare (type (double-float) z y x))
    (prog ((w 0.0d0) (xabs 0.0d0) (yabs 0.0d0) (zabs 0.0d0) (dlapy3 0.0d0))
      (declare (type (double-float) dlapy3 zabs yabs xabs w)) (setf xabs (abs x))
      (setf yabs (abs y)) (setf zabs (abs z)) (setf w (max xabs yabs zabs))
      (cond ((= w zero) (setf dlapy3 (+ xabs yabs zabs)))
            (t
             (setf dlapy3
               (* w
                 (f2cl-lib:fsqrt
                  (+ (expt (/ xabs w) 2) (expt (/ yabs w) 2) (expt (/ zabs w) 2)))))))
      (go end_label) end_label (return (values dlapy3 nil nil nil)))))

```

dlqatr LAPACK

— dlqatr.input —

```

)set break resume
)sys rm -f dlqatr.output
)spool dlqatr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlqatr.help —

```
=====
dlaqtr examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLAQTR - the real quasi-triangular system $op(T)*p = scale*c$, if LREAL = .TRUE

SYNOPSIS

```
SUBROUTINE DLAQTR( LTRAN, LREAL, N, T, LDT, B, W, SCALE, X, WORK, INFO
 )
```

LOGICAL LREAL, LTRAN

INTEGER INFO, LDT, N

DOUBLE PRECISION SCALE, W

DOUBLE PRECISION B(*), T(LDT, *), WORK(*), X(*)

Purpose

```
=====
```

DLAQTR solves the real quasi-triangular system

$$op(T)*p = scale*c, \quad \text{if } LREAL = .TRUE.$$

or the complex quasi-triangular systems

$$op(T + iB)*(p+iq) = scale*(c+id), \quad \text{if } LREAL = .FALSE.$$

in real arithmetic, where T is upper quasi-triangular.

If LREAL = .FALSE., then the first diagonal block of T must be 1 by 1, B is the specially structured matrix

$$B = \begin{bmatrix} b(1) & b(2) & \dots & b(n) \\ & w & & \\ & & w & \\ & & & . \\ & & & & w \end{bmatrix}$$

$op(A) = A$ or A' , A' denotes the conjugate transpose of matrix A.

On input, $X = \begin{bmatrix} c \\ d \end{bmatrix}$. On output, $X = \begin{bmatrix} p \\ q \end{bmatrix}$.

This subroutine is designed for the condition number estimation in routine DTRSNA.

Arguments

=====

LTRAN (input) LOGICAL
On entry, LTRAN specifies the option of conjugate transpose:
= .FALSE., op(T+i*B) = T+i*B,
= .TRUE., op(T+i*B) = (T+i*B)'.

LREAL (input) LOGICAL
On entry, LREAL specifies the input matrix structure:
= .FALSE., the input is complex
= .TRUE., the input is real

N (input) INTEGER
On entry, N specifies the order of T+i*B. N >= 0.

T (input) DOUBLE PRECISION array, dimension (LDT,N)
On entry, T contains a matrix in Schur canonical form.
If LREAL = .FALSE., then the first diagonal block of T must be 1 by 1.

LDT (input) INTEGER
The leading dimension of the matrix T. LDT >= max(1,N).

B (input) DOUBLE PRECISION array, dimension (N)
On entry, B contains the elements to form the matrix B as described above.
If LREAL = .TRUE., B is not referenced.

W (input) DOUBLE PRECISION
On entry, W is the diagonal element of the matrix B.
If LREAL = .TRUE., W is not referenced.

SCALE (output) DOUBLE PRECISION
On exit, SCALE is the scale factor.

X (input/output) DOUBLE PRECISION array, dimension (2*N)
On entry, X contains the right hand side of the system.
On exit, X is overwritten by the solution.

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
On exit, INFO is set to
0: successful exit.
1: the some diagonal 1 by 1 block has been perturbed by

a small number SMIN to keep nonsingularity.
 2: the some diagonal 2 by 2 block has been perturbed by
 a small number in DLALN2 to keep nonsingularity.
 NOTE: In the interests of speed, this routine does not
 check the inputs for errors.

— dlaqtr.f —

```

SUBROUTINE DLAQTR( LTRAN, LREAL, N, T, LDT, B, W, SCALE, X, WORK,
$                INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
LOGICAL          LREAL, LTRAN
INTEGER          INFO, LDT, N
DOUBLE PRECISION SCALE, W
*
*  .. Array Arguments ..
DOUBLE PRECISION B( * ), T( LDT, * ), WORK( * ), X( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
LOGICAL          NOTRAN
INTEGER          I, IERR, J, J1, J2, JNEXT, K, N1, N2
DOUBLE PRECISION BIGNUM, EPS, REC, SCALOC, SI, SMIN, SMINW,
$               SMLNUM, SR, TJJ, TMP, XJ, XMAX, XNORM, Z
*
*  ..
*  .. Local Arrays ..
DOUBLE PRECISION D( 2, 2 ), V( 2, 2 )
*
*  ..
*  .. External Functions ..
INTEGER          IDAMAX
DOUBLE PRECISION DASUM, DDOT, DLAMCH, DLANGE
EXTERNAL         IDAMAX, DASUM, DDOT, DLAMCH, DLANGE
*
*  ..
*  .. External Subroutines ..

```

```

      EXTERNAL          DAXPY, DLADIV, DLALN2, DSCAL
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX
*
*   ..
*   .. Executable Statements ..
*
*   Do not test the input parameters for errors
*
      NOTRAN = .NOT.LTRAN
      INFO = 0
*
*   Quick return if possible
*
      IF( N.EQ.0 )
$    RETURN
*
*   Set constants to control overflow
*
      EPS = DLAMCH( 'P' )
      SMLNUM = DLAMCH( 'S' ) / EPS
      BIGNUM = ONE / SMLNUM
*
      XNORM = DLANGE( 'M', N, N, T, LDT, D )
      IF( .NOT.LREAL )
$    XNORM = MAX( XNORM, ABS( W ), DLANGE( 'M', N, 1, B, N, D ) )
      SMIN = MAX( SMLNUM, EPS*XNORM )
*
*   Compute 1-norm of each column of strictly upper triangular
*   part of T to control overflow in triangular solver.
*
      WORK( 1 ) = ZERO
      DO 10 J = 2, N
        WORK( J ) = DASUM( J-1, T( 1, J ), 1 )
10    CONTINUE
*
      IF( .NOT.LREAL ) THEN
        DO 20 I = 2, N
          WORK( I ) = WORK( I ) + ABS( B( I ) )
20    CONTINUE
      END IF
*
      N2 = 2*N
      N1 = N
      IF( .NOT.LREAL )
$    N1 = N2
      K = IDAMAX( N1, X, 1 )
      XMAX = ABS( X( K ) )
      SCALE = ONE
*

```

```

      IF( XMAX.GT.BIGNUM ) THEN
        SCALE = BIGNUM / XMAX
        CALL DSCAL( N1, SCALE, X, 1 )
        XMAX = BIGNUM
      END IF
*
      IF( LREAL ) THEN
*
        IF( NOTRAN ) THEN
*
          Solve T*p = scale*c
*
          JNEXT = N
          DO 30 J = N, 1, -1
            IF( J.GT.JNEXT )
$              GO TO 30
              J1 = J
              J2 = J
              JNEXT = J - 1
              IF( J.GT.1 ) THEN
                IF( T( J, J-1 ).NE.ZERO ) THEN
                  J1 = J - 1
                  JNEXT = J - 2
                END IF
              END IF
            END IF
*
            IF( J1.EQ.J2 ) THEN
*
              Meet 1 by 1 diagonal block
*
              Scale to avoid overflow when computing
              x(j) = b(j)/T(j,j)
*
              XJ = ABS( X( J1 ) )
              TJJ = ABS( T( J1, J1 ) )
              TMP = T( J1, J1 )
              IF( TJJ.LT.SMIN ) THEN
                TMP = SMIN
                TJJ = SMIN
                INFO = 1
              END IF
*
              IF( XJ.EQ.ZERO )
$                GO TO 30
*
              IF( TJJ.LT.ONE ) THEN
                IF( XJ.GT.BIGNUM*TJJ ) THEN
                  REC = ONE / XJ
                  CALL DSCAL( N, REC, X, 1 )
                  SCALE = SCALE*REC

```

```

        XMAX = XMAX*REC
      END IF
    END IF
    X( J1 ) = X( J1 ) / TMP
    XJ = ABS( X( J1 ) )

*
*      Scale x if necessary to avoid overflow when adding a
*      multiple of column j1 of T.
*

    IF( XJ.GT.ONE ) THEN
      REC = ONE / XJ
      IF( WORK( J1 ).GT.( BIGNUM-XMAX )*REC ) THEN
        CALL DSCAL( N, REC, X, 1 )
        SCALE = SCALE*REC
      END IF
    END IF
    IF( J1.GT.1 ) THEN
      CALL DAXPY( J1-1, -X( J1 ), T( 1, J1 ), 1, X, 1 )
      K = IDAMAX( J1-1, X, 1 )
      XMAX = ABS( X( K ) )
    END IF

*
*      ELSE
*
*      Meet 2 by 2 diagonal block
*
*      Call 2 by 2 linear system solve, to take
*      care of possible overflow by scaling factor.
*

    D( 1, 1 ) = X( J1 )
    D( 2, 1 ) = X( J2 )
    CALL DLALN2( .FALSE., 2, 1, SMIN, ONE, T( J1, J1 ),
$              LDT, ONE, ONE, D, 2, ZERO, ZERO, V, 2,
$              SCALOC, XNORM, IERR )
    IF( IERR.NE.0 )
$      INFO = 2

*

    IF( SCALOC.NE.ONE ) THEN
      CALL DSCAL( N, SCALOC, X, 1 )
      SCALE = SCALE*SCALOC
    END IF
    X( J1 ) = V( 1, 1 )
    X( J2 ) = V( 2, 1 )

*
*      Scale V(1,1) (= X(J1)) and/or V(2,1) (=X(J2))
*      to avoid overflow in updating right-hand side.
*

    XJ = MAX( ABS( V( 1, 1 ) ), ABS( V( 2, 1 ) ) )
    IF( XJ.GT.ONE ) THEN
      REC = ONE / XJ

```

```

      IF( MAX( WORK( J1 ), WORK( J2 ) ).GT.
          ( BIGNUM-XMAX )*REC ) THEN
          CALL DSCAL( N, REC, X, 1 )
          SCALE = SCALE*REC
      END IF
END IF

Update right-hand side

IF( J1.GT.1 ) THEN
    CALL DAXPY( J1-1, -X( J1 ), T( 1, J1 ), 1, X, 1 )
    CALL DAXPY( J1-1, -X( J2 ), T( 1, J2 ), 1, X, 1 )
    K = IDAMAX( J1-1, X, 1 )
    XMAX = ABS( X( K ) )
END IF

END IF

30      CONTINUE

ELSE

    Solve T'*p = scale*c

    JNEXT = 1
    DO 40 J = 1, N
        IF( J.LT.JNEXT )
            GO TO 40
        J1 = J
        J2 = J
        JNEXT = J + 1
        IF( J.LT.N ) THEN
            IF( T( J+1, J ).NE.ZERO ) THEN
                J2 = J + 1
                JNEXT = J + 2
            END IF
        END IF

        IF( J1.EQ.J2 ) THEN

            1 by 1 diagonal block

            Scale if necessary to avoid overflow in forming the
            right-hand side element by inner product.

            XJ = ABS( X( J1 ) )
            IF( XMAX.GT.ONE ) THEN
                REC = ONE / XMAX
                IF( WORK( J1 ).GT.( BIGNUM-XJ )*REC ) THEN
                    CALL DSCAL( N, REC, X, 1 )

```



```

        SCALE = SCALE*REC
        XMAX = XMAX*REC
    END IF
END IF
*
X( J1 ) = X( J1 ) - DDOT( J1-1, T( 1, J1 ), 1, X, 1 )
*
XJ = ABS( X( J1 ) )
TJJ = ABS( T( J1, J1 ) )
TMP = T( J1, J1 )
IF( TJJ.LT.SMIN ) THEN
    TMP = SMIN
    TJJ = SMIN
    INFO = 1
END IF
*
IF( TJJ.LT.ONE ) THEN
    IF( XJ.GT.BIGNUM*TJJ ) THEN
        REC = ONE / XJ
        CALL DSCAL( N, REC, X, 1 )
        SCALE = SCALE*REC
        XMAX = XMAX*REC
    END IF
END IF
X( J1 ) = X( J1 ) / TMP
XMAX = MAX( XMAX, ABS( X( J1 ) ) )
*
ELSE
*
*      2 by 2 diagonal block
*
*      Scale if necessary to avoid overflow in forming the
*      right-hand side elements by inner product.
*
XJ = MAX( ABS( X( J1 ) ), ABS( X( J2 ) ) )
IF( XMAX.GT.ONE ) THEN
    REC = ONE / XMAX
    IF( MAX( WORK( J2 ), WORK( J1 ) ).GT.( BIGNUM-XJ ) *
$      REC ) THEN
        CALL DSCAL( N, REC, X, 1 )
        SCALE = SCALE*REC
        XMAX = XMAX*REC
    END IF
END IF
*
D( 1, 1 ) = X( J1 ) - DDOT( J1-1, T( 1, J1 ), 1, X,
$      1 )
D( 2, 1 ) = X( J2 ) - DDOT( J1-1, T( 1, J2 ), 1, X,
$      1 )
*

```

```

CALL DLALN2( .TRUE., 2, 1, SMIN, ONE, T( J1, J1 ),
$           LDT, ONE, ONE, D, 2, ZERO, ZERO, V, 2,
$           SCALOC, XNORM, IERR )
IF( IERR.NE.0 )
$   INFO = 2
*
IF( SCALOC.NE.ONE ) THEN
  CALL DSCAL( N, SCALOC, X, 1 )
  SCALE = SCALE*SCALOC
END IF
X( J1 ) = V( 1, 1 )
X( J2 ) = V( 2, 1 )
XMAX = MAX( ABS( X( J1 ) ), ABS( X( J2 ) ), XMAX )
*
END IF
40  CONTINUE
END IF
*
ELSE
*
SMINW = MAX( EPS*ABS( W ), SMIN )
IF( NOTRAN ) THEN
*
  Solve (T + iB)*(p+iq) = c+id
*
  JNEXT = N
  DO 70 J = N, 1, -1
    IF( J.GT.JNEXT )
$      GO TO 70
    J1 = J
    J2 = J
    JNEXT = J - 1
    IF( J.GT.1 ) THEN
      IF( T( J, J-1 ).NE.ZERO ) THEN
        J1 = J - 1
        JNEXT = J - 2
      END IF
    END IF
  END IF
*
  IF( J1.EQ.J2 ) THEN
*
    1 by 1 diagonal block
*
    Scale if necessary to avoid overflow in division
*
    Z = W
    IF( J1.EQ.1 )
$      Z = B( 1 )
    XJ = ABS( X( J1 ) ) + ABS( X( N+J1 ) )
    TJJ = ABS( T( J1, J1 ) ) + ABS( Z )

```

```

      TMP = T( J1, J1 )
      IF( TJJ.LT.SMINW ) THEN
        TMP = SMINW
        TJJ = SMINW
        INFO = 1
      END IF
*
      IF( XJ.EQ.ZERO )
$        GO TO 70
*
      IF( TJJ.LT.ONE ) THEN
        IF( XJ.GT.BIGNUM*TJJ ) THEN
          REC = ONE / XJ
          CALL DSCAL( N2, REC, X, 1 )
          SCALE = SCALE*REC
          XMAX = XMAX*REC
        END IF
      END IF
      CALL DLADIV( X( J1 ), X( N+J1 ), TMP, Z, SR, SI )
      X( J1 ) = SR
      X( N+J1 ) = SI
      XJ = ABS( X( J1 ) ) + ABS( X( N+J1 ) )
*
*      Scale x if necessary to avoid overflow when adding a
*      multiple of column j1 of T.
*
      IF( XJ.GT.ONE ) THEN
        REC = ONE / XJ
        IF( WORK( J1 ).GT.( BIGNUM-XMAX )*REC ) THEN
          CALL DSCAL( N2, REC, X, 1 )
          SCALE = SCALE*REC
        END IF
      END IF
*
      IF( J1.GT.1 ) THEN
        CALL DAXPY( J1-1, -X( J1 ), T( 1, J1 ), 1, X, 1 )
        CALL DAXPY( J1-1, -X( N+J1 ), T( 1, J1 ), 1,
$          X( N+1 ), 1 )
*
        X( 1 ) = X( 1 ) + B( J1 )*X( N+J1 )
        X( N+1 ) = X( N+1 ) - B( J1 )*X( J1 )
*
        XMAX = ZERO
        DO 50 K = 1, J1 - 1
          XMAX = MAX( XMAX, ABS( X( K ) )+
$            ABS( X( K+N ) ) )
50      CONTINUE
      END IF
*
      ELSE

```

```

*
*      Meet 2 by 2 diagonal block
*
      D( 1, 1 ) = X( J1 )
      D( 2, 1 ) = X( J2 )
      D( 1, 2 ) = X( N+J1 )
      D( 2, 2 ) = X( N+J2 )
      CALL DLALN2( .FALSE., 2, 2, SMINW, ONE, T( J1, J1 ),
$              LDT, ONE, ONE, D, 2, ZERO, -W, V, 2,
$              SCALOC, XNORM, IERR )
      IF( IERR.NE.0 )
$          INFO = 2
*
      IF( SCALOC.NE.ONE ) THEN
          CALL DSCAL( 2*N, SCALOC, X, 1 )
          SCALE = SCALOC*SCALE
      END IF
      X( J1 ) = V( 1, 1 )
      X( J2 ) = V( 2, 1 )
      X( N+J1 ) = V( 1, 2 )
      X( N+J2 ) = V( 2, 2 )
*
*      Scale X(J1), .... to avoid overflow in
*      updating right hand side.
*
      XJ = MAX( ABS( V( 1, 1 ) )+ABS( V( 1, 2 ) ),
$          ABS( V( 2, 1 ) )+ABS( V( 2, 2 ) ) )
      IF( XJ.GT.ONE ) THEN
          REC = ONE / XJ
          IF( MAX( WORK( J1 ), WORK( J2 ) ).GT.
$              ( BIGNUM-XMAX )*REC ) THEN
              CALL DSCAL( N2, REC, X, 1 )
              SCALE = SCALE*REC
          END IF
      END IF
*
*      Update the right-hand side.
*
      IF( J1.GT.1 ) THEN
          CALL DAXPY( J1-1, -X( J1 ), T( 1, J1 ), 1, X, 1 )
          CALL DAXPY( J1-1, -X( J2 ), T( 1, J2 ), 1, X, 1 )
*
          CALL DAXPY( J1-1, -X( N+J1 ), T( 1, J1 ), 1,
$              X( N+1 ), 1 )
          CALL DAXPY( J1-1, -X( N+J2 ), T( 1, J2 ), 1,
$              X( N+1 ), 1 )
*
          X( 1 ) = X( 1 ) + B( J1 )*X( N+J1 ) +
$              B( J2 )*X( N+J2 )
          X( N+1 ) = X( N+1 ) - B( J1 )*X( J1 ) -

```

```

$          B( J2 ) * X( J2 )
*
      XMAX = ZERO
      DO 60 K = 1, J1 - 1
        XMAX = MAX( ABS( X( K ) ) + ABS( X( K+N ) ),
$          XMAX )
60      CONTINUE
      END IF
*
      END IF
70      CONTINUE
*
      ELSE
*
*      Solve (T + iB)'*(p+iq) = c+id
*
      JNEXT = 1
      DO 80 J = 1, N
        IF( J.LT.JNEXT )
$          GO TO 80
        J1 = J
        J2 = J
        JNEXT = J + 1
        IF( J.LT.N ) THEN
          IF( T( J+1, J ).NE.ZERO ) THEN
            J2 = J + 1
            JNEXT = J + 2
          END IF
        END IF
      END IF
*
      IF( J1.EQ.J2 ) THEN
*
*      1 by 1 diagonal block
*
*      Scale if necessary to avoid overflow in forming the
*      right-hand side element by inner product.
*
      XJ = ABS( X( J1 ) ) + ABS( X( J1+N ) )
      IF( XMAX.GT.ONE ) THEN
        REC = ONE / XMAX
        IF( WORK( J1 ).GT.( BIGNUM-XJ ) * REC ) THEN
          CALL DSCAL( N2, REC, X, 1 )
          SCALE = SCALE * REC
          XMAX = XMAX * REC
        END IF
      END IF
*
      X( J1 ) = X( J1 ) - DDOT( J1-1, T( 1, J1 ), 1, X, 1 )
      X( N+J1 ) = X( N+J1 ) - DDOT( J1-1, T( 1, J1 ), 1,
$      X( N+1 ), 1 )

```

```

IF( J1.GT.1 ) THEN
    X( J1 ) = X( J1 ) - B( J1 )*X( N+1 )
    X( N+J1 ) = X( N+J1 ) + B( J1 )*X( 1 )
END IF
XJ = ABS( X( J1 ) ) + ABS( X( J1+N ) )

*
Z = W
IF( J1.EQ.1 )
$    Z = B( 1 )

*
*
*   Scale if necessary to avoid overflow in
*   complex division
*

TJJ = ABS( T( J1, J1 ) ) + ABS( Z )
TMP = T( J1, J1 )
IF( TJJ.LT.SMINW ) THEN
    TMP = SMINW
    TJJ = SMINW
    INFO = 1
END IF

*

IF( TJJ.LT.ONE ) THEN
    IF( XJ.GT.BIGNUM*TJJ ) THEN
        REC = ONE / XJ
        CALL DSCAL( N2, REC, X, 1 )
        SCALE = SCALE*REC
        XMAX = XMAX*REC
    END IF
END IF
CALL DLADIV( X( J1 ), X( N+J1 ), TMP, -Z, SR, SI )
X( J1 ) = SR
X( J1+N ) = SI
XMAX = MAX( ABS( X( J1 ) )+ABS( X( J1+N ) ), XMAX )

*

ELSE

*
*
*   2 by 2 diagonal block
*
*   Scale if necessary to avoid overflow in forming the
*   right-hand side element by inner product.
*

XJ = MAX( ABS( X( J1 ) )+ABS( X( N+J1 ) ),
$       ABS( X( J2 ) )+ABS( X( N+J2 ) ) )
IF( XMAX.GT.ONE ) THEN
    REC = ONE / XMAX
    IF( MAX( WORK( J1 ), WORK( J2 ) ).GT.
$       ( BIGNUM-XJ ) / XMAX ) THEN
        CALL DSCAL( N2, REC, X, 1 )
        SCALE = SCALE*REC
        XMAX = XMAX*REC
    END IF
END IF

```

```

      END IF
    END IF
*
      D( 1, 1 ) = X( J1 ) - DDOT( J1-1, T( 1, J1 ), 1, X,
$          1 )
      D( 2, 1 ) = X( J2 ) - DDOT( J1-1, T( 1, J2 ), 1, X,
$          1 )
      D( 1, 2 ) = X( N+J1 ) - DDOT( J1-1, T( 1, J1 ), 1,
$          X( N+1 ), 1 )
      D( 2, 2 ) = X( N+J2 ) - DDOT( J1-1, T( 1, J2 ), 1,
$          X( N+1 ), 1 )
      D( 1, 1 ) = D( 1, 1 ) - B( J1 )*X( N+1 )
      D( 2, 1 ) = D( 2, 1 ) - B( J2 )*X( N+1 )
      D( 1, 2 ) = D( 1, 2 ) + B( J1 )*X( 1 )
      D( 2, 2 ) = D( 2, 2 ) + B( J2 )*X( 1 )
*
      CALL DLALN2( .TRUE., 2, 2, SMINW, ONE, T( J1, J1 ),
$          LDT, ONE, ONE, D, 2, ZERO, W, V, 2,
$          SCALOC, XNORM, IERR )
      IF( IERR.NE.0 )
$          INFO = 2
*
      IF( SCALOC.NE.ONE ) THEN
        CALL DSCAL( N2, SCALOC, X, 1 )
        SCALE = SCALOC*SCALE
      END IF
      X( J1 ) = V( 1, 1 )
      X( J2 ) = V( 2, 1 )
      X( N+J1 ) = V( 1, 2 )
      X( N+J2 ) = V( 2, 2 )
      XMAX = MAX( ABS( X( J1 ) )+ABS( X( N+J1 ) ),
$          ABS( X( J2 ) )+ABS( X( N+J2 ) ), XMAX )
*
      END IF
*
80      CONTINUE
*
      END IF
*
      END IF
*
      RETURN
*
      End of DLAQTR
*
      END

```

— LAPACK dlaqtr —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlaqtr (ltran lreal n t$ ldt b w scale x work info)
    (declare (type (double-float) scale w)
              (type (simple-array double-float (*)) work x b t$)
              (type fixnum info ldt n)
              (type (member t nil) lreal ltran))
    (f2cl-lib:with-multi-array-data
      ((t$ double-float t$-%data% t$-%offset%)
       (b double-float b-%data% b-%offset%)
       (x double-float x-%data% x-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((d (make-array 4 :element-type 'double-float))
              (v (make-array 4 :element-type 'double-float)) (bignum 0.0)
              (eps 0.0) (rec 0.0) (scaloc 0.0) (si 0.0) (smin 0.0) (sminw 0.0)
              (smlnum 0.0) (sr 0.0) (tjj 0.0) (tmp 0.0) (xj 0.0) (xmax 0.0)
              (xnorm 0.0) (z 0.0) (i 0) (ierr 0) (j 0) (j1 0) (j2 0) (jnext 0)
              (k 0) (n1 0) (n2 0) (notran nil))
              (declare (type (simple-array double-float (4)) d v)
                        (type (double-float) bignum eps rec scaloc si smin sminw
                               smlnum sr tjj tmp xj xmax xnorm z)
                        (type fixnum i ierr j j1 j2 jnext k n1 n2)
                        (type (member t nil) notran))
              (setf notran (not ltran))
              (setf info 0)
              (if (= n 0) (go end_label))
              (setf eps (dlamch "P"))
              (setf smlnum (/ (dlamch "S") eps))
              (setf bignum (/ one smlnum))
              (setf xnorm (dlange "M" n n t$ ldt d))
              (if (not lreal)
                  (setf xnorm (max xnorm (abs w) (dlange "M" n 1 b n d))))
              (setf smin (max smlnum (* eps xnorm)))
              (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) zero)
              (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                            ((> j n) nil)
                (tagbody
                  (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
                        (dasum (f2cl-lib:int-sub j 1)
                              (f2cl-lib:array-slice t$
                                                       double-float
                                                       (1 j)
                                                       ((1 ldt) (1 *)))
                        1))))
              1))))
  (cond
    ((not lreal)

```



```

(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
    (+ (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
      (abs
        (f2cl-lib:fref b-%data% (i) ((1 *)) b-%offset%))))))tpd
(setf n2 (f2cl-lib:int-mul 2 n))
(setf n1 n)
(if (not lreal) (setf n1 n2))
(setf k (idamax n1 x 1))
(setf xmax (abs (f2cl-lib:fref x-%data% (k) ((1 *)) x-%offset%)))
(setf scale one)
(cond
  (> xmax bignum)
  (setf scale (/ bignum xmax))
  (dscal n1 scale x 1)
  (setf xmax bignum)))
(cond
  (lreal
    (cond
      (notran
        (setf jnext n)
        (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
          (> j 1) nil)
        (tagbody
          (if (> j jnext) (go label30))
          (setf j1 j)
          (setf j2 j)
          (setf jnext (f2cl-lib:int-sub j 1))
          (cond
            (> j 1)
            (cond
              (/=
                (f2cl-lib:fref t$
                  (j
                    (f2cl-lib:int-add j
                      (f2cl-lib:int-sub
                        1)))
                    ((1 ldt) (1 *)))
                zero)
              (setf j1 (f2cl-lib:int-sub j 1))
              (setf jnext (f2cl-lib:int-sub j 2))))))
          (cond
            (= j1 j2)
            (setf xj
              (abs
                (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
            (setf tj
              (abs

```

```

(f2cl-lib:fref t$-%data%
  (j1 j1)
  ((1 ldt) (1 *)))
t$-%offset%)))
(setf tmp
  (f2cl-lib:fref t$-%data%
    (j1 j1)
    ((1 ldt) (1 *)))
    t$-%offset%)))
(cond
  ((< tjj smin)
   (setf tmp smin)
   (setf tjj smin)
   (setf info 1)))
(if (= xj zero) (go label30))
(cond
  ((< tjj one)
   (cond
    ((> xj (* bignum tjj))
     (setf rec (/ one xj))
     (dscal n rec x 1)
     (setf scale (* scale rec))
     (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
  (/
    (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
    tmp))
(setf xj
  (abs
    (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
(cond
  ((> xj one)
   (setf rec (/ one xj))
   (cond
    ((> (f2cl-lib:fref work (j1) ((1 *)))
      (* (+ bignum (- xmax)) rec))
     (dscal n rec x 1)
     (setf scale (* scale rec))))))
(cond
  ((> j1 1)
   (daxpy (f2cl-lib:int-sub j1 1)
    (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (f2cl-lib:array-slice t$
        double-float
        (1 j1)
        ((1 ldt) (1 *)))
        1 x 1)
    (setf k (idamax (f2cl-lib:int-sub j1 1) x 1))
    (setf xmax
      (abs

```

```

(f2cl-lib:fref x-%data%
(k)
((1 *))
x-%offset%))))))
(t
(setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
(f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
(setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
(f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10 var-11 var-12 var-13 var-14
var-15 var-16 var-17)
(dlaln2 nil 2 1 smin one
(f2cl-lib:array-slice t$
double-float
(j1 j1)
((1 ldt) (1 *)))
ldt one one d 2 zero zero v 2 scaloc xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-7 var-8 var-9 var-10 var-11
var-12 var-13 var-14))
(setf scaloc var-15)
(setf xnorm var-16)
(setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
((/= scaloc one)
(dscal n scaloc x 1)
(setf scale (* scale scaloc))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
(f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
(f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf xj
(max (abs (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(abs (f2cl-lib:fref v (2 1) ((1 2) (1 2)))))
(cond
(> xj one)
(setf rec (/ one xj))
(cond
(>
(max (f2cl-lib:fref work (j1) ((1 *)))
(f2cl-lib:fref work (j2) ((1 *)))
(* (+ bignum (- xmax)) rec))
(dscal n rec x 1)
(setf scale (* scale rec))))))
(cond
(> j1 1)
(daxpy (f2cl-lib:int-sub j1 1)

```

```

(- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
(f2cl-lib:array-slice t$
  double-float
  (1 j1)
  ((1 ldt) (1 *)))
1 x 1)
(daxpy (f2cl-lib:int-sub j1 1)
  (- (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
  (f2cl-lib:array-slice t$
    double-float
    (1 j2)
    ((1 ldt) (1 *)))
  1 x 1)
(setf k (idamax (f2cl-lib:int-sub j1 1) x 1))
(setf xmax
  (abs
    (f2cl-lib:fref x-%data%
      (k)
      ((1 *))
      x-%offset%))))))
label30)))
(t
  (setf jnext 1)
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (if (< j jnext) (go label40))
      (setf j1 j)
      (setf j2 j)
      (setf jnext (f2cl-lib:int-add j 1))
      (cond
        ((< j n)
          (cond
            ((/=
              (f2cl-lib:fref t$
                ((f2cl-lib:int-add j 1) j)
                ((1 ldt) (1 *)))
              zero)
              (setf j2 (f2cl-lib:int-add j 1))
              (setf jnext (f2cl-lib:int-add j 2))))))
          (cond
            ((= j1 j2)
              (setf xj
                (abs
                  (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
              (cond
                ((> xmax one)
                  (setf rec (/ one xmax))
                  (cond
                    ((> (f2cl-lib:fref work (j1) ((1 *)))

```

```

        (* (+ bignum (- xj)) rec))
(dscal n rec x 1)
(setf scale (* scale rec))
(setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (-
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        (ddot (f2cl-lib:int-sub j1 1)
              (f2cl-lib:array-slice t$
                                    double-float
                                    (1 j1)
                                    ((1 ldt) (1 *)))
              1 x 1)))
(setf xj
      (abs
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)))
(setf tjj
      (abs
        (f2cl-lib:fref t$-%data%
                      (j1 j1)
                      ((1 ldt) (1 *))
                      t$-%offset%)))
(setf tmp
      (f2cl-lib:fref t$-%data%
                    (j1 j1)
                    ((1 ldt) (1 *))
                    t$-%offset%)))
(cond
 ((< tjj smin)
  (setf tmp smin)
  (setf tjj smin)
  (setf info 1)))
(cond
 ((< tjj one)
  (cond
   ((> xj (* bignum tjj))
    (setf rec (/ one xj))
    (dscal n rec x 1)
    (setf scale (* scale rec))
    (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (/
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        tmp))
(setf xmax
      (max xmax
            (abs
              (f2cl-lib:fref x-%data%
                            (j1)
                            ((1 *))

```

```

                                x-%offset%))))))
(t
  (setf xj
    (max
      (abs
        (f2cl-lib:fref x-%data%
          (j1)
          ((1 *))
          x-%offset%))
      (abs
        (f2cl-lib:fref x-%data%
          (j2)
          ((1 *))
          x-%offset%))))))
(cond
  ((> xmax one)
    (setf rec (/ one xmax))
    (cond
      ((>
        (max (f2cl-lib:fref work (j2) ((1 *))
              (f2cl-lib:fref work (j1) ((1 *))))
        (* (+ bignum (- xj)) rec))
        (dscal n rec x 1)
        (setf scale (* scale rec))
        (setf xmax (* xmax rec))))))
  (setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
    (-
      (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (ddot (f2cl-lib:int-sub j1 1)
        (f2cl-lib:array-slice t$
          double-float
          (1 j1)
          ((1 ldt) (1 *)))
          1 x 1)))
  (setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
    (-
      (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
      (ddot (f2cl-lib:int-sub j1 1)
        (f2cl-lib:array-slice t$
          double-float
          (1 j2)
          ((1 ldt) (1 *)))
          1 x 1)))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12 var-13 var-14
     var-15 var-16 var-17)
    (dlaln2 t 2 1 smin one
      (f2cl-lib:array-slice t$
        double-float

```

```

                                (j1 j1)
                                ((1 ldt) (1 *)))
      ldt one one d 2 zero zero v 2 scaloc xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9 var-10 var-11
              var-12 var-13 var-14))
(setf scaloc var-15)
(setf xnorm var-16)
(setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
  ((/= scaloc one)
   (dscal n scaloc x 1)
   (setf scale (* scale scaloc))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%
      (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%
      (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf xmax
  (max
    (abs
      (f2cl-lib:fref x-%data%
                    (j1)
                    ((1 *))
                    x-%offset%))
    (abs
      (f2cl-lib:fref x-%data%
                    (j2)
                    ((1 *))
                    x-%offset%))
    xmax))))
label40))))))
(t
  (setf sminw (max (* eps (abs w)) smin))
  (cond
    (notran
      (setf jnext n)
      (f2cl-lib:fd0 (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                    (> j 1) nil)
      (tagbody
        (if (> j jnext) (go label70))
        (setf j1 j)
        (setf j2 j)
        (setf jnext (f2cl-lib:int-sub j 1))
        (cond
          ((> j 1)
           (cond
            ((/=
              (f2cl-lib:fref t$
                            (j

```

```

                                (f2cl-lib:int-add j
                                (f2cl-lib:int-sub
                                  1)))
                                ((1 ldt) (1 *)))
                                zero)
  (setf j1 (f2cl-lib:int-sub j 1))
  (setf jnext (f2cl-lib:int-sub j 2))))))
(cond
  ((= j1 j2)
   (setf z w)
   (if (= j1 1)
       (setf z
              (f2cl-lib:fref b-%data%
                             (1)
                             ((1 *))
                             b-%offset%)))
       (setf xj
              (+
               (abs
                (f2cl-lib:fref x-%data%
                               (j1)
                               ((1 *))
                               x-%offset%))
               (abs
                (f2cl-lib:fref x-%data%
                               ((f2cl-lib:int-add n j1))
                               ((1 *))
                               x-%offset%))))))
       (setf tjj
              (+
               (abs
                (f2cl-lib:fref t$-%data%
                               (j1 j1)
                               ((1 ldt) (1 *))
                               t$-%offset%))
               (abs z)))
       (setf tmp
              (f2cl-lib:fref t$-%data%
                             (j1 j1)
                             ((1 ldt) (1 *))
                             t$-%offset%)))
   (cond
     ((< tjj sminw)
      (setf tmp sminw)
      (setf tjj sminw)
      (setf info 1)))
     (if (= xj zero) (go label70))
     (cond
       ((< tjj one)
        (cond

```



```

      (> xj (* bignum tjj))
      (setf rec (/ one xj))
      (dscal n2 rec x 1)
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
  (dladiv
    (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))
      x-%offset%)
    tmp z sr si)
  (declare (ignore var-0 var-1 var-2 var-3))
  (setf sr var-4)
  (setf si var-5))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%) sr)
(setf (f2cl-lib:fref x-%data%
  ((f2cl-lib:int-add n j1))
  ((1 *))
  x-%offset%)
  si)
(setf xj
  (+
    (abs
      (f2cl-lib:fref x-%data%
        (j1)
        ((1 *))
        x-%offset%))
    (abs
      (f2cl-lib:fref x-%data%
        ((f2cl-lib:int-add n j1))
        ((1 *))
        x-%offset%))))))
(cond
  (> xj one)
  (setf rec (/ one xj))
  (cond
    (> (f2cl-lib:fref work (j1) ((1 *)))
      (* (+ bignum (- xmax)) rec))
    (dscal n2 rec x 1)
    (setf scale (* scale rec))))))
(cond
  (> j1 1)
  (daxpy (f2cl-lib:int-sub j1 1)
    (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (f2cl-lib:array-slice t$
        double-float
        (1 j1)
        ((1 ldt) (1 *)))

```

```

1 x 1)
(daxpy (f2cl-lib:int-sub j1 1)
(-
  (f2cl-lib:fref x-%data%
    ((f2cl-lib:int-add n j1))
    ((1 *))
    x-%offset%))
(f2cl-lib:array-slice t$
  double-float
  (1 j1)
  ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice x
  double-float
  ((+ n 1))
  ((1 *)))
1)
(setf (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%)
(+
  (f2cl-lib:fref x-%data%
    (1)
    ((1 *))
    x-%offset%)
  (*
    (f2cl-lib:fref b-%data%
      (j1)
      ((1 *))
      b-%offset%)
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))
      x-%offset%))))))
(setf (f2cl-lib:fref x-%data%
  ((f2cl-lib:int-add n 1))
  ((1 *))
  x-%offset%)
(-
  (f2cl-lib:fref x-%data%
    ((f2cl-lib:int-add n 1))
    ((1 *))
    x-%offset%)
  (*
    (f2cl-lib:fref b-%data%
      (j1)
      ((1 *))
      b-%offset%)
    (f2cl-lib:fref x-%data%
      (j1)
      ((1 *))
      x-%offset%))))))

```

```

(setf xmax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add j1
      (f2cl-lib:int-sub
        1)))
    nil)
(tagbody
  (setf xmax
    (max xmax
      (+
        (abs
          (f2cl-lib:fref x-%data%
            (k)
            ((1 *))
            x-%offset%))
        (abs
          (f2cl-lib:fref x-%data%
            ((f2cl-lib:int-add k
              n))
            ((1 *))
            x-%offset%))))))
(t
  (setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
    (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%))
  (setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
    (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%))
  (setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))
      x-%offset%))
  (setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j2))
      ((1 *))
      x-%offset%))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12 var-13 var-14
     var-15 var-16 var-17)
    (dlaln2 nil 2 2 sminw one
      (f2cl-lib:array-slice t$
        double-float
        (j1 j1)
        ((1 ldt) (1 *)))
      ldt one one d 2 zero (- w) v 2 scaloc xnorm ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6 var-7 var-8 var-9 var-10 var-11
      var-12 var-13 var-14)))

```

```

      (setf scaloc var-15)
      (setf xnorm var-16)
      (setf ierr var-17))
    (if (/= ierr 0) (setf info 2))
    (cond
      ((/= scaloc one)
        (dscal (f2cl-lib:int-mul 2 n) scaloc x 1)
        (setf scale (* scaloc scale))))
    (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
          (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
    (setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
          (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
    (setf (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))
                        ((1 *))
                        x-%offset%)
          (f2cl-lib:fref v (1 2) ((1 2) (1 2))))
    (setf (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j2))
                        ((1 *))
                        x-%offset%)
          (f2cl-lib:fref v (2 2) ((1 2) (1 2))))
    (setf xj
      (max
        (+ (abs (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
          (abs (f2cl-lib:fref v (1 2) ((1 2) (1 2))))
        (+ (abs (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
          (abs (f2cl-lib:fref v (2 2) ((1 2) (1 2)))))))
    (cond
      ((> xj one)
        (setf rec (/ one xj))
        (cond
          ((>
            (max (f2cl-lib:fref work (j1) ((1 *)))
                  (f2cl-lib:fref work (j2) ((1 *))))
            (* (+ bignum (- xmax)) rec))
            (dscal n2 rec x 1)
            (setf scale (* scale rec))))))
      (cond
        ((> j1 1)
          (daxpy (f2cl-lib:int-sub j1 1)
            (- (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
              (f2cl-lib:array-slice t$
                                    double-float
                                    (1 j1)
                                    ((1 ldt) (1 *)))
            1 x 1)
          (daxpy (f2cl-lib:int-sub j1 1)
            (- (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
              (f2cl-lib:array-slice t$

```

```

                                double-float
                                (1 j2)
                                ((1 ldt) (1 *)))
1 x 1)
(daxpy (f2cl-lib:int-sub j1 1)
(-
  (f2cl-lib:fref x-%data%
    ((f2cl-lib:int-add n j1))
    ((1 *))
    x-%offset%))
(f2cl-lib:array-slice t$
  double-float
  (1 j1)
  ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice x
  double-float
  ((+ n 1))
  ((1 *)))
1)
(daxpy (f2cl-lib:int-sub j1 1)
(-
  (f2cl-lib:fref x-%data%
    ((f2cl-lib:int-add n j2))
    ((1 *))
    x-%offset%))
(f2cl-lib:array-slice t$
  double-float
  (1 j2)
  ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice x
  double-float
  ((+ n 1))
  ((1 *)))
1)
(setf (f2cl-lib:fref x-%data% (1) ((1 *)) x-%offset%)
(+
  (f2cl-lib:fref x-%data%
    (1)
    ((1 *))
    x-%offset%)
  (*
    (f2cl-lib:fref b-%data%
      (j1)
      ((1 *))
      b-%offset%)
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))

```

```

                                x-%offset%))
(*
  (f2cl-lib:fref b-%data%
                (j2)
                ((1 *))
                b-%offset%)
  (f2cl-lib:fref x-%data%
                ((f2cl-lib:int-add n j2))
                ((1 *))
                x-%offset%))))
(setf (f2cl-lib:fref x-%data%
                    ((f2cl-lib:int-add n 1))
                    ((1 *))
                    x-%offset%))

(-
  (f2cl-lib:fref x-%data%
                ((f2cl-lib:int-add n 1))
                ((1 *))
                x-%offset%)

  (*
    (f2cl-lib:fref b-%data%
                  (j1)
                  ((1 *))
                  b-%offset%)
    (f2cl-lib:fref x-%data%
                  (j1)
                  ((1 *))
                  x-%offset%))

  (*
    (f2cl-lib:fref b-%data%
                  (j2)
                  ((1 *))
                  b-%offset%)
    (f2cl-lib:fref x-%data%
                  (j2)
                  ((1 *))
                  x-%offset%))))))
(setf xmax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add j1
                      (f2cl-lib:int-sub
                        1)))
    nil)
(tagbody
  (setf xmax
    (max
      (+
        (abs
          (f2cl-lib:fref x-%data%

```

```

(k)
((1 *))
x-%offset%))

(abs
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add k n))
((1 *))
x-%offset%)))

xmax)))))))))

label70)))

(t
(setf jnext 1)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j n) nil)
(tagbody
  (if (< j jnext) (go label80))
  (setf j1 j)
  (setf j2 j)
  (setf jnext (f2cl-lib:int-add j 1))
  (cond
    ((< j n)
     (cond
      ((/=
        (f2cl-lib:fref t$
          ((f2cl-lib:int-add j 1) j)
          ((1 ldt) (1 *)))
         zero)
       (setf j2 (f2cl-lib:int-add j 1))
       (setf jnext (f2cl-lib:int-add j 2))))))
    (cond
      ((= j1 j2)
       (setf xj
        (+
         (abs
          (f2cl-lib:fref x-%data%
            (j1)
            ((1 *))
            x-%offset%))
         (abs
          (f2cl-lib:fref x-%data%
            ((f2cl-lib:int-add j1 n))
            ((1 *))
            x-%offset%))))))
      (cond
        ((> xmax one)
         (setf rec (/ one xmax))
         (cond
          ((> (f2cl-lib:fref work (j1) ((1 *)))
              (* (+ bignum (- xj)) rec))
           (dscal n2 rec x 1)

```

```

(setf scale (* scale rec))
(setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
  (-
    (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
    (ddot (f2cl-lib:int-sub j1 1)
      (f2cl-lib:array-slice t$
        double-float
        (1 j1)
        ((1 ldt) (1 *)))
      1 x 1)))
(setf (f2cl-lib:fref x-%data%
  ((f2cl-lib:int-add n j1))
  ((1 *))
  x-%offset%)
  (-
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))
      x-%offset%)
    (ddot (f2cl-lib:int-sub j1 1)
      (f2cl-lib:array-slice t$
        double-float
        (1 j1)
        ((1 ldt) (1 *)))
      1
      (f2cl-lib:array-slice x
        double-float
        ((+ n 1))
        ((1 *)))
      1)))
(cond
  ((> j1 1)
    (setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (-
        (f2cl-lib:fref x-%data%
          (j1)
          ((1 *))
          x-%offset%)
        (*
          (f2cl-lib:fref b-%data%
            (j1)
            ((1 *))
            b-%offset%)
          (f2cl-lib:fref x-%data%
            ((f2cl-lib:int-add n 1))
            ((1 *))
            x-%offset%))))))
(setf (f2cl-lib:fref x-%data%
  ((f2cl-lib:int-add n j1))
  ((1 *))
  x-%offset%)
  (-
    (f2cl-lib:fref x-%data%
      ((f2cl-lib:int-add n j1))
      ((1 *))
      x-%offset%)
    (*
      (f2cl-lib:fref b-%data%
        (j1)
        ((1 *))
        b-%offset%)
      (f2cl-lib:fref x-%data%
        ((f2cl-lib:int-add n 1))
        ((1 *))
        x-%offset%))))))

```



```

((1 *))
x-%offset%)
(+
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add n j1))
((1 *))
x-%offset%)
(*
(f2cl-lib:fref b-%data%
(j1)
((1 *))
b-%offset%)
(f2cl-lib:fref x-%data%
(1)
((1 *))
x-%offset%))))))
(setf xj
(+
(abs
(f2cl-lib:fref x-%data%
(j1)
((1 *))
x-%offset%))
(abs
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add j1 n))
((1 *))
x-%offset%))))))
(setf z w)
(if (= j1 1)
(setf z
(f2cl-lib:fref b-%data%
(1)
((1 *))
b-%offset%)))
(setf tjj
(+
(abs
(f2cl-lib:fref t$-%data%
(j1 j1)
((1 ldt) (1 *))
t$-%offset%))
(abs z)))
(setf tmp
(f2cl-lib:fref t$-%data%
(j1 j1)
((1 ldt) (1 *))
t$-%offset%))
(cond
(< tjj sminw)

```

```
(setf tmp sminw)
(setf tjj sminw)
(setf info 1)))
(cond
 (((< tjj one)
  (cond
   (> xj (* bignum tjj))
   (setf rec (/ one xj))
   (dscal n2 rec x 1)
   (setf scale (* scale rec))
   (setf xmax (* xmax rec))))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
 (dladiv
  (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
  (f2cl-lib:fref x-%data%
                  ((f2cl-lib:int-add n j1))
                  ((1 *))
                  x-%offset%)
  tmp (- z) sr si)
 (declare (ignore var-0 var-1 var-2 var-3))
 (setf sr var-4)
 (setf si var-5))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%) sr)
(setf (f2cl-lib:fref x-%data%
                    ((f2cl-lib:int-add j1 n))
                    ((1 *))
                    x-%offset%)
      si)
(setf xmax
  (max
   (+
    (abs
     (f2cl-lib:fref x-%data%
                     (j1)
                     ((1 *))
                     x-%offset%))
    (abs
     (f2cl-lib:fref x-%data%
                     ((f2cl-lib:int-add j1 n))
                     ((1 *))
                     x-%offset%))))
    xmax)))
(t
 (setf xj
  (max
   (+
    (abs
     (f2cl-lib:fref x-%data%
                     (j1)
                     ((1 *)
```

```

                                x-%offset%))
      (abs
        (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))
                        ((1 *))
                        x-%offset%)))
    (+
      (abs
        (f2cl-lib:fref x-%data%
                        (j2)
                        ((1 *))
                        x-%offset%))
      (abs
        (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j2))
                        ((1 *))
                        x-%offset%))))))
(cond
  ((> xmax one)
   (setf rec (/ one xmax))
   (cond
    (>
     (max (f2cl-lib:fref work (j1) ((1 *)))
           (f2cl-lib:fref work (j2) ((1 *)))
           (f2cl-lib:f2cl/ (+ bignum (- xj)) xmax))
     (dscal n2 rec x 1)
     (setf scale (* scale rec))
     (setf xmax (* xmax rec))))))
(setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
      (-
        (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
        (ddot (f2cl-lib:int-sub j1 1)
              (f2cl-lib:array-slice t$
                                    double-float
                                    (1 j1)
                                    ((1 ldt) (1 *)))
              1 x 1)))
(setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
      (-
        (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
        (ddot (f2cl-lib:int-sub j1 1)
              (f2cl-lib:array-slice t$
                                    double-float
                                    (1 j2)
                                    ((1 ldt) (1 *)))
              1 x 1)))
(setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
      (-
        (f2cl-lib:fref x-%data%
                        ((f2cl-lib:int-add n j1))

```

```

                                ((1 *))
                                x-%offset%)
(ddot (f2cl-lib:int-sub j1 1)
(f2cl-lib:array-slice t$
                                double-float
                                (1 j1)
                                ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice x
                                double-float
                                ((+ n 1))
                                ((1 *)))
1)))
(setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
(-
(f2cl-lib:fref x-%data%
                                ((f2cl-lib:int-add n j2))
                                ((1 *))
                                x-%offset%)
(ddot (f2cl-lib:int-sub j1 1)
(f2cl-lib:array-slice t$
                                double-float
                                (1 j2)
                                ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice x
                                double-float
                                ((+ n 1))
                                ((1 *)))
1)))
(setf (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
(- (f2cl-lib:fref d (1 1) ((1 2) (1 2)))
(*
(f2cl-lib:fref b-%data%
(j1)
((1 *))
b-%offset%)
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add n 1))
((1 *))
x-%offset%))))
(setf (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
(- (f2cl-lib:fref d (2 1) ((1 2) (1 2)))
(*
(f2cl-lib:fref b-%data%
(j2)
((1 *))
b-%offset%)
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add n 1))

```

```

                                ((1 *))
                                x-%offset%))))
(setf (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
      (+ (f2cl-lib:fref d (1 2) ((1 2) (1 2)))
         (*
           (f2cl-lib:fref b-%data%
                           (j1)
                           ((1 *))
                           b-%offset%)
           (f2cl-lib:fref x-%data%
                           (1)
                           ((1 *))
                           x-%offset%))))))
(setf (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
      (+ (f2cl-lib:fref d (2 2) ((1 2) (1 2)))
         (*
           (f2cl-lib:fref b-%data%
                           (j2)
                           ((1 *))
                           b-%offset%)
           (f2cl-lib:fref x-%data%
                           (1)
                           ((1 *))
                           x-%offset%))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14
   var-15 var-16 var-17)
  (dlaln2 t 2 2 sminw one
    (f2cl-lib:array-slice t$
                          double-float
                          (j1 j1)
                          ((1 ldt) (1 *)))
    ldt one one d 2 zero w v 2 scaloc xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9 var-10 var-11
                    var-12 var-13 var-14))
  (setf scaloc var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(if (/= ierr 0) (setf info 2))
(cond
 ((/= scaloc one)
  (dscal n2 scaloc x 1)
  (setf scale (* scaloc scale))))
(setf (f2cl-lib:fref x-%data% (j1) ((1 *)) x-%offset%)
      (f2cl-lib:fref v (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data% (j2) ((1 *)) x-%offset%)
      (f2cl-lib:fref v (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%

```

```

((f2cl-lib:int-add n j1))
((1 *))
x-%offset%)
(f2cl-lib:fref v (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref x-%data%
((f2cl-lib:int-add n j2))
((1 *))
x-%offset%)
(f2cl-lib:fref v (2 2) ((1 2) (1 2))))
(setf xmax
(max
(+
(abs
(f2cl-lib:fref x-%data%
(j1)
((1 *))
x-%offset%))
(abs
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add n j1))
((1 *))
x-%offset%)))
(+
(abs
(f2cl-lib:fref x-%data%
(j2)
((1 *))
x-%offset%))
(abs
(f2cl-lib:fref x-%data%
((f2cl-lib:int-add n j2))
((1 *))
x-%offset%)))
xmax))))
label80))))))
end_label
(return (values nil nil nil nil nil nil nil scale nil nil info))))))

```

dlarfb LAPACK

— dlarfb.input —

```

)set break resume
)sys rm -f dlarfb.output

```

```
)spool dlarfb.output
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

—————
— dlarfb.help —

```
=====
dlarfb examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLARFB - a real block reflector H or its transpose H' to a real m by n matrix C, from either the left or the right

SYNOPSIS

```
SUBROUTINE DLARFB( SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV, T,
                  LDT, C, LDC, WORK, LDWORK )
```

CHARACTER DIRECT, SIDE, STOREV, TRANS

INTEGER K, LDC, LDT, LDV, LDWORK, M, N

DOUBLE PRECISION C(LDC, *), T(LDT, *), V(LDV, *),
 WORK(LDWORK, *)

Purpose

```
=====
```

DLARFB applies a real block reflector H or its transpose H' to a real m by n matrix C, from either the left or the right.

Arguments

```
=====
```

SIDE (input) CHARACTER*1
 = 'L': apply H or H' from the Left
 = 'R': apply H or H' from the Right

TRANS (input) CHARACTER*1

```

= 'N': apply H (No transpose)
= 'T': apply H' (Transpose)

DIRECT  (input) CHARACTER*1
Indicates how H is formed from a product of elementary
reflectors
= 'F': H = H(1) H(2) . . . H(k) (Forward)
= 'B': H = H(k) . . . H(2) H(1) (Backward)

STOREV  (input) CHARACTER*1
Indicates how the vectors which define the elementary
reflectors are stored:
= 'C': Columnwise
= 'R': Rowwise

M        (input) INTEGER
The number of rows of the matrix C.

N        (input) INTEGER
The number of columns of the matrix C.

K        (input) INTEGER
The order of the matrix T (= the number of elementary
reflectors whose product defines the block reflector).

V        (input) DOUBLE PRECISION array, dimension
                (LDV,K) if STOREV = 'C'
                (LDV,M) if STOREV = 'R' and SIDE = 'L'
                (LDV,N) if STOREV = 'R' and SIDE = 'R'
The matrix V. See further details.

LDV      (input) INTEGER
The leading dimension of the array V.
If STOREV = 'C' and SIDE = 'L', LDV >= max(1,M);
if STOREV = 'C' and SIDE = 'R', LDV >= max(1,N);
if STOREV = 'R', LDV >= K.

T        (input) DOUBLE PRECISION array, dimension (LDT,K)
The triangular k by k matrix T in the representation of the
block reflector.

LDT      (input) INTEGER
The leading dimension of the array T. LDT >= K.

C        (input/output) DOUBLE PRECISION array, dimension (LDC,N)
On entry, the m by n matrix C.
On exit, C is overwritten by H*C or H'*C or C*H or C*H'.

LDC      (input) INTEGER
The leading dimension of the array C. LDA >= max(1,M).

```


WORK (workspace) DOUBLE PRECISION array, dimension (LDWORK,K)

LDWORK (input) INTEGER

The leading dimension of the array WORK.

If SIDE = 'L', LDWORK \geq max(1,N);

if SIDE = 'R', LDWORK \geq max(1,M).

— dlarfb.f —

```

      SUBROUTINE DLARFB( SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV,
$                      T, LDT, C, LDC, WORK, LDWORK )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
      CHARACTER          DIRECT, SIDE, STOREV, TRANS
      INTEGER            K, LDC, LDT, LDV, LDWORK, M, N
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION   C( LDC, * ), T( LDT, * ), V( LDV, * ),
$                      WORK( LDWORK, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
      CHARACTER          TRANST
      INTEGER            I, J
*
*  ..
*
*  .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*
*  ..
*
*  .. External Subroutines ..
      EXTERNAL           DCOPY, DGEMM, DTRMM
*
*  ..
*
*  .. Executable Statements ..
*

```

```

*      Quick return if possible
*
      IF( M.LE.0 .OR. N.LE.0 )
$     RETURN
*
      IF( LSAME( TRANS, 'N' ) ) THEN
          TRANST = 'T'
      ELSE
          TRANST = 'N'
      END IF
*
      IF( LSAME( STOREV, 'C' ) ) THEN
*
          IF( LSAME( DIRECT, 'F' ) ) THEN
*
              Let  V = ( V1 )    (first K rows)
                     ( V2 )
*              where V1 is unit lower triangular.
*
              IF( LSAME( SIDE, 'L' ) ) THEN
*
                  Form  H * C  or  H' * C  where  C = ( C1 )
                                                         ( C2 )
*
                  W := C' * V  =  (C1'*V1 + C2'*V2)  (stored in WORK)
*
                  W := C1'
*
                  DO 10 J = 1, K
                      CALL DCOPY( N, C( J, 1 ), LDC, WORK( 1, J ), 1 )
10                 CONTINUE
*
                  W := W * V1
*
                  CALL DTRMM( 'Right', 'Lower', 'No transpose', 'Unit', N,
$                      K, ONE, V, LDV, WORK, LDWORK )
                  IF( M.GT.K ) THEN
*
                      W := W + C2'*V2
*
                      CALL DGEMM( 'Transpose', 'No transpose', N, K, M-K,
$                      ONE, C( K+1, 1 ), LDC, V( K+1, 1 ), LDV,
$                      ONE, WORK, LDWORK )
                  END IF
*
                  W := W * T'  or  W * T
*
                  CALL DTRMM( 'Right', 'Upper', TRANST, 'Non-unit', N, K,
$                      ONE, T, LDT, WORK, LDWORK )
*

```

```

*          C := C - V * W'
*
*          IF( M.GT.K ) THEN
*
*              C2 := C2 - V2 * W'
*
*              CALL DGEMM( 'No transpose', 'Transpose', M-K, N, K,
$                  -ONE, V( K+1, 1 ), LDV, WORK, LDWORK, ONE,
$                  C( K+1, 1 ), LDC )
*
*              END IF
*
*              W := W * V1'
*
*              CALL DTRMM( 'Right', 'Lower', 'Transpose', 'Unit', N, K,
$                  ONE, V, LDV, WORK, LDWORK )
*
*              C1 := C1 - W'
*
*              DO 30 J = 1, K
*                  DO 20 I = 1, N
*                      C( J, I ) = C( J, I ) - WORK( I, J )
20                  CONTINUE
30              CONTINUE
*
*          ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*              Form C * H or C * H' where C = ( C1 C2 )
*
*              W := C * V = (C1*V1 + C2*V2) (stored in WORK)
*
*              W := C1
*
*              DO 40 J = 1, K
*                  CALL DCOPY( M, C( 1, J ), 1, WORK( 1, J ), 1 )
40              CONTINUE
*
*              W := W * V1
*
*              CALL DTRMM( 'Right', 'Lower', 'No transpose', 'Unit', M,
$                  K, ONE, V, LDV, WORK, LDWORK )
*
*              IF( N.GT.K ) THEN
*
*                  W := W + C2 * V2
*
*                  CALL DGEMM( 'No transpose', 'No transpose', M, K, N-K,
$                      ONE, C( 1, K+1 ), LDC, V( K+1, 1 ), LDV,
$                      ONE, WORK, LDWORK )
*
*                  END IF
*
*              W := W * T or W * T'

```

```

*
*      CALL DTRMM( 'Right', 'Upper', TRANS, 'Non-unit', M, K,
$          ONE, T, LDT, WORK, LDWORK )
*
*      C := C - W * V'
*
*      IF( N.GT.K ) THEN
*
*          C2 := C2 - W * V2'
*
*          CALL DGEMM( 'No transpose', 'Transpose', M, N-K, K,
$              -ONE, WORK, LDWORK, V( K+1, 1 ), LDV, ONE,
$              C( 1, K+1 ), LDC )
*
*          END IF
*
*      W := W * V1'
*
*      CALL DTRMM( 'Right', 'Lower', 'Transpose', 'Unit', M, K,
$          ONE, V, LDV, WORK, LDWORK )
*
*      C1 := C1 - W
*
*      DO 60 J = 1, K
*          DO 50 I = 1, M
*              C( I, J ) = C( I, J ) - WORK( I, J )
50          CONTINUE
60      CONTINUE
*      END IF
*
*      ELSE
*
*      Let V = ( V1 )
*              ( V2 ) (last K rows)
*      where V2 is unit upper triangular.
*
*      IF( LSAME( SIDE, 'L' ) ) THEN
*
*          Form H * C or H' * C where C = ( C1 )
*                                          ( C2 )
*
*          W := C' * V = (C1'*V1 + C2'*V2) (stored in WORK)
*
*          W := C2'
*
*          DO 70 J = 1, K
*              CALL DCOPY( N, C( M-K+J, 1 ), LDC, WORK( 1, J ), 1 )
70          CONTINUE
*
*          W := W * V2
*

```

```

      CALL DTRMM( 'Right', 'Upper', 'No transpose', 'Unit', N,
$           K, ONE, V( M-K+1, 1 ), LDV, WORK, LDWORK )
      IF( M.GT.K ) THEN
*
*           W := W + C1*V1
*
*           CALL DGEMM( 'Transpose', 'No transpose', N, K, M-K,
$           ONE, C, LDC, V, LDV, ONE, WORK, LDWORK )
      END IF
*
*           W := W * T' or W * T
*
*           CALL DTRMM( 'Right', 'Lower', TRANST, 'Non-unit', N, K,
$           ONE, T, LDT, WORK, LDWORK )
*
*           C := C - V * W'
*
*           IF( M.GT.K ) THEN
*
*               C1 := C1 - V1 * W'
*
*               CALL DGEMM( 'No transpose', 'Transpose', M-K, N, K,
$               -ONE, V, LDV, WORK, LDWORK, ONE, C, LDC )
      END IF
*
*           W := W * V2'
*
*           CALL DTRMM( 'Right', 'Upper', 'Transpose', 'Unit', N, K,
$           ONE, V( M-K+1, 1 ), LDV, WORK, LDWORK )
*
*           C2 := C2 - W'
*
      DO 90 J = 1, K
          DO 80 I = 1, N
              C( M-K+J, I ) = C( M-K+J, I ) - WORK( I, J )
80          CONTINUE
90          CONTINUE
*
      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*           Form C * H or C * H' where C = ( C1 C2 )
*
*           W := C * V = (C1*V1 + C2*V2) (stored in WORK)
*
*           W := C2
*
      DO 100 J = 1, K
          CALL DCOPY( M, C( 1, N-K+J ), 1, WORK( 1, J ), 1 )
100         CONTINUE
*

```

```

*          W := W * V2
*
*          CALL DTRMM( 'Right', 'Upper', 'No transpose', 'Unit', M,
$              K, ONE, V( N-K+1, 1 ), LDV, WORK, LDWORK )
*          IF( N.GT.K ) THEN
*
*              W := W + C1 * V1
*
*              CALL DGEMM( 'No transpose', 'No transpose', M, K, N-K,
$              ONE, C, LDC, V, LDV, ONE, WORK, LDWORK )
*          END IF
*
*          W := W * T  or  W * T'
*
*          CALL DTRMM( 'Right', 'Lower', TRANS, 'Non-unit', M, K,
$              ONE, T, LDT, WORK, LDWORK )
*
*          C := C - W * V'
*
*          IF( N.GT.K ) THEN
*
*              C1 := C1 - W * V1'
*
*              CALL DGEMM( 'No transpose', 'Transpose', M, N-K, K,
$              -ONE, WORK, LDWORK, V, LDV, ONE, C, LDC )
*          END IF
*
*          W := W * V2'
*
*          CALL DTRMM( 'Right', 'Upper', 'Transpose', 'Unit', M, K,
$              ONE, V( N-K+1, 1 ), LDV, WORK, LDWORK )
*
*          C2 := C2 - W
*
*          DO 120 J = 1, K
*              DO 110 I = 1, M
*                  C( I, N-K+J ) = C( I, N-K+J ) - WORK( I, J )
110                  CONTINUE
120                  CONTINUE
*          END IF
*          END IF
*
*          ELSE IF( LSAME( STOREV, 'R' ) ) THEN
*
*              IF( LSAME( DIRECT, 'F' ) ) THEN
*
*                  Let V = ( V1 V2 )    (V1: first K columns)
*                  where V1 is unit upper triangular.
*
*                  IF( LSAME( SIDE, 'L' ) ) THEN

```

```

*
*      Form  $H * C$  or  $H' * C$  where  $C = \begin{pmatrix} C1 \\ C2 \end{pmatrix}$ 
*
*       $W := C' * V' = (C1' * V1' + C2' * V2')$  (stored in WORK)
*
*       $W := C1'$ 
*
      DO 130 J = 1, K
130      CALL DCOPY( N, C( J, 1 ), LDC, WORK( 1, J ), 1 )
      CONTINUE
*
*       $W := W * V1'$ 
*
      CALL DTRMM( 'Right', 'Upper', 'Transpose', 'Unit', N, K,
$              ONE, V, LDV, WORK, LDWORK )
      IF( M.GT.K ) THEN
*
*           $W := W + C2' * V2'$ 
*
          CALL DGEMM( 'Transpose', 'Transpose', N, K, M-K, ONE,
$                  C( K+1, 1 ), LDC, V( 1, K+1 ), LDV, ONE,
$                  WORK, LDWORK )
          END IF
*
*       $W := W * T'$  or  $W * T$ 
*
      CALL DTRMM( 'Right', 'Upper', TRANST, 'Non-unit', N, K,
$              ONE, T, LDT, WORK, LDWORK )
*
*       $C := C - V' * W'$ 
*
      IF( M.GT.K ) THEN
*
*           $C2 := C2 - V2' * W'$ 
*
          CALL DGEMM( 'Transpose', 'Transpose', M-K, N, K, -ONE,
$                  V( 1, K+1 ), LDV, WORK, LDWORK, ONE,
$                  C( K+1, 1 ), LDC )
          END IF
*
*       $W := W * V1$ 
*
      CALL DTRMM( 'Right', 'Upper', 'No transpose', 'Unit', N,
$              K, ONE, V, LDV, WORK, LDWORK )
*
*       $C1 := C1 - W'$ 
*
      DO 150 J = 1, K
          DO 140 I = 1, N

```

```

          C( J, I ) = C( J, I ) - WORK( I, J )
140      CONTINUE
150      CONTINUE
*
      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*          Form  $C * H$  or  $C * H'$  where  $C = ( C_1 \ C_2 )$ 
*
*           $W := C * V' = (C_1 * V_1' + C_2 * V_2')$  (stored in WORK)
*
*           $W := C_1$ 
*
      DO 160 J = 1, K
          CALL DCOPY( M, C( 1, J ), 1, WORK( 1, J ), 1 )
160      CONTINUE
*
*           $W := W * V_1'$ 
*
*          CALL DTRMM( 'Right', 'Upper', 'Transpose', 'Unit', M, K,
$              ONE, V, LDV, WORK, LDWORK )
*          IF( N.GT.K ) THEN
*
*               $W := W + C_2 * V_2'$ 
*
*              CALL DGEMM( 'No transpose', 'Transpose', M, K, N-K,
$                  ONE, C( 1, K+1 ), LDC, V( 1, K+1 ), LDV,
$                  ONE, WORK, LDWORK )
*
*          END IF
*
*           $W := W * T$  or  $W * T'$ 
*
*          CALL DTRMM( 'Right', 'Upper', TRANS, 'Non-unit', M, K,
$              ONE, T, LDT, WORK, LDWORK )
*
*           $C := C - W * V$ 
*
*          IF( N.GT.K ) THEN
*
*               $C_2 := C_2 - W * V_2$ 
*
*              CALL DGEMM( 'No transpose', 'No transpose', M, N-K, K,
$                  -ONE, WORK, LDWORK, V( 1, K+1 ), LDV, ONE,
$                  C( 1, K+1 ), LDC )
*
*          END IF
*
*           $W := W * V_1$ 
*
*          CALL DTRMM( 'Right', 'Upper', 'No transpose', 'Unit', M,
$              K, ONE, V, LDV, WORK, LDWORK )
*

```



```

*          C1 := C1 - W
*
*          DO 180 J = 1, K
*            DO 170 I = 1, M
*              C( I, J ) = C( I, J ) - WORK( I, J )
170          CONTINUE
180          CONTINUE
*
*          END IF
*
*          ELSE
*
*          Let V = ( V1 V2 )    (V2: last K columns)
*          where V2 is unit lower triangular.
*
*          IF( LSAME( SIDE, 'L' ) ) THEN
*
*          Form H * C or H' * C where C = ( C1 )
*                                           ( C2 )
*
*          W := C' * V' = (C1'*V1' + C2'*V2') (stored in WORK)
*
*          W := C2'
*
*          DO 190 J = 1, K
*            CALL DCOPY( N, C( M-K+J, 1 ), LDC, WORK( 1, J ), 1 )
190          CONTINUE
*
*          W := W * V2'
*
*          CALL DTRMM( 'Right', 'Lower', 'Transpose', 'Unit', N, K,
$              ONE, V( 1, M-K+1 ), LDV, WORK, LDWORK )
*          IF( M.GT.K ) THEN
*
*              W := W + C1'*V1'
*
*          CALL DGEMM( 'Transpose', 'Transpose', N, K, M-K, ONE,
$              C, LDC, V, LDV, ONE, WORK, LDWORK )
*          END IF
*
*          W := W * T' or W * T
*
*          CALL DTRMM( 'Right', 'Lower', TRANST, 'Non-unit', N, K,
$              ONE, T, LDT, WORK, LDWORK )
*
*          C := C - V' * W'
*
*          IF( M.GT.K ) THEN
*
*              C1 := C1 - V1' * W'

```

```

*
*      CALL DGEMM( 'Transpose', 'Transpose', M-K, N, K, -ONE,
$          V, LDV, WORK, LDWORK, ONE, C, LDC )
*      END IF
*
*      W := W * V2
*
*      CALL DTRMM( 'Right', 'Lower', 'No transpose', 'Unit', N,
$          K, ONE, V( 1, M-K+1 ), LDV, WORK, LDWORK )
*
*      C2 := C2 - W'
*
*      DO 210 J = 1, K
*          DO 200 I = 1, N
*              C( M-K+J, I ) = C( M-K+J, I ) - WORK( I, J )
200          CONTINUE
210      CONTINUE
*
*      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*          Form C * H or C * H' where C = ( C1 C2 )
*
*          W := C * V' = (C1*V1' + C2*V2') (stored in WORK)
*
*          W := C2
*
*          DO 220 J = 1, K
*              CALL DCOPY( M, C( 1, N-K+J ), 1, WORK( 1, J ), 1 )
220          CONTINUE
*
*          W := W * V2'
*
*          CALL DTRMM( 'Right', 'Lower', 'Transpose', 'Unit', M, K,
$              ONE, V( 1, N-K+1 ), LDV, WORK, LDWORK )
*          IF( N.GT.K ) THEN
*
*              W := W + C1 * V1'
*
*              CALL DGEMM( 'No transpose', 'Transpose', M, K, N-K,
$                  ONE, C, LDC, V, LDV, ONE, WORK, LDWORK )
*          END IF
*
*          W := W * T or W * T'
*
*          CALL DTRMM( 'Right', 'Lower', TRANS, 'Non-unit', M, K,
$              ONE, T, LDT, WORK, LDWORK )
*
*          C := C - W * V
*
*          IF( N.GT.K ) THEN

```

```

*
*          C1 := C1 - W * V1
*
*          CALL DGEMM( 'No transpose', 'No transpose', M, N-K, K,
$              -ONE, WORK, LDWORK, V, LDV, ONE, C, LDC )
*          END IF
*
*          W := W * V2
*
*          CALL DTRMM( 'Right', 'Lower', 'No transpose', 'Unit', M,
$              K, ONE, V( 1, N-K+1 ), LDV, WORK, LDWORK )
*
*          C1 := C1 - W
*
*          DO 240 J = 1, K
*              DO 230 I = 1, M
*                  C( I, N-K+J ) = C( I, N-K+J ) - WORK( I, J )
230          CONTINUE
240          CONTINUE
*
*          END IF
*
*          END IF
*          END IF
*
*          RETURN
*
*          End of DLARFB
*
*          END

```

— LAPACK dlarfb —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dlarfb (side trans direct storev m n k v ldv t$ ldt c ldc work ldwork)
    (declare (type (simple-array double-float (*)) work c t$ v)
      (type fixnum ldwork ldc ldt ldv k n m)
      (type character storev direct trans side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (direct character direct-%data% direct-%offset%)
       (storev character storev-%data% storev-%offset%)
       (v double-float v-%data% v-%offset%)
       (t$ double-float t$-%data% t$-%offset%))

```

```

(c double-float c-%data% c-%offset%)
(work double-float work-%data% work-%offset%))
(prog ((i 0) (j 0)
      (transt
        (make-array '(1) :element-type 'character :initial-element #\ )))
(declare (type fixnum i j)
          (type (simple-array character (1)) transt))
(if (or (<= m 0) (<= n 0)) (go end_label))
(cond
  ((char-equal trans #\N)
   (f2cl-lib:f2cl-set-string transt "T" (string 1)))
  (t
   (f2cl-lib:f2cl-set-string transt "N" (string 1))))
(cond
  ((char-equal storev #\C)
   (cond
    ((char-equal direct #\F)
     (cond
      ((char-equal side #\L)
       (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                     (> j k) nil)
       (tagbody
        (dcopy n
               (f2cl-lib:array-slice c
                                     double-float
                                     (j 1)
                                     ((1 ldc) (1 *)))
               ldc
               (f2cl-lib:array-slice work
                                     double-float
                                     (1 j)
                                     ((1 ldwork) (1 *)))
               1)))
      (dtrmm "Right" "Lower" "No transpose" "Unit" n k one v ldv
              work ldwork))
    (cond
     (> m k)
     (dgemm "Transpose" "No transpose" n k
            (f2cl-lib:int-sub m k) one
            (f2cl-lib:array-slice c
                                  double-float
                                  ((+ k 1) 1)
                                  ((1 ldc) (1 *)))
            ldc
            (f2cl-lib:array-slice v
                                  double-float
                                  ((+ k 1) 1)
                                  ((1 ldv) (1 *)))
            ldv one work ldwork)))
    (dtrmm "Right" "Upper" transt "Non-unit" n k one t$ ldt work
           work ldwork)))

```

```

ldwork)
(cond
  (> m k)
  (dgemm "No transpose" "Transpose" (f2cl-lib:int-sub m k) n
    k (- one)
    (f2cl-lib:array-slice v
      double-float
      ((+ k 1) 1)
      ((1 ldv) (1 *)))
    ldv work ldwork one
    (f2cl-lib:array-slice c
      double-float
      ((+ k 1) 1)
      ((1 ldc) (1 *)))
    ldc)))
(dtrmm "Right" "Lower" "Transpose" "Unit" n k one v ldv work
ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (j i)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j i)
          ((1 ldc) (1 *))
          c-%offset%)
        (f2cl-lib:fref work-%data%
          (i j)
          ((1 ldwork) (1 *))
          work-%offset%))))))
((char-equal side #\R)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (dcopy m
    (f2cl-lib:array-slice c
      double-float
      (1 j)
      ((1 ldc) (1 *)))
    1
    (f2cl-lib:array-slice work
      double-float
      (1 j)
      ((1 ldwork) (1 *)))

```

```

1)))
(dtrmm "Right" "Lower" "No transpose" "Unit" m k one v ldv
work ldwork)
(cond
  (> n k)
  (dgemm "No transpose" "No transpose" m k
    (f2cl-lib:int-sub n k) one
    (f2cl-lib:array-slice c
      double-float
      (1 (f2cl-lib:int-add k 1))
      ((1 ldc) (1 *)))
    ldc
    (f2cl-lib:array-slice v
      double-float
      ((+ k 1) 1)
      ((1 ldv) (1 *)))
    ldv one work ldwork)))
(dtrmm "Right" "Upper" trans "Non-unit" m k one t$ ldt work
ldwork)
(cond
  (> n k)
  (dgemm "No transpose" "Transpose" m (f2cl-lib:int-sub n k)
    k (- one) work ldwork
    (f2cl-lib:array-slice v
      double-float
      ((+ k 1) 1)
      ((1 ldv) (1 *)))
    ldv one
    (f2cl-lib:array-slice c
      double-float
      (1 (f2cl-lib:int-add k 1))
      ((1 ldc) (1 *)))
    ldc)))
(dtrmm "Right" "Lower" "Transpose" "Unit" m k one v ldv work
ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (i j)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (i j)
          ((1 ldc) (1 *))
          c-%offset%)

```

```

                                (f2cl-lib:fref work-%data%
                                (i j)
                                ((1 ldwork) (1 *))
                                work-%offset%)))))))))
(t
  (cond
    ((char-equal side #\L)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j k) nil)
        (tagbody
          (dcopy n
            (f2cl-lib:array-slice c
              double-float
              ((+ m (f2cl-lib:int-sub k) j) 1)
              ((1 ldc) (1 *)))
            ldc
            (f2cl-lib:array-slice work
              double-float
              (1 j)
              ((1 ldwork) (1 *)))
            1)))
      (dtrmm "Right" "Upper" "No transpose" "Unit" n k one
        (f2cl-lib:array-slice v
          double-float
          ((+ m (f2cl-lib:int-sub k) 1) 1)
          ((1 ldv) (1 *)))
        ldv work ldwork)
      (cond
        ((> m k)
          (dgemm "Transpose" "No transpose" n k
            (f2cl-lib:int-sub m k) one c ldc v ldv one work ldwork)))
      (dtrmm "Right" "Lower" transt "Non-unit" n k one t$ ldt work
        ldwork)
      (cond
        ((> m k)
          (dgemm "No transpose" "Transpose" (f2cl-lib:int-sub m k) n
            k (- one) v ldv work ldwork one c ldc)))
      (dtrmm "Right" "Upper" "Transpose" "Unit" n k one
        (f2cl-lib:array-slice v
          double-float
          ((+ m (f2cl-lib:int-sub k) 1) 1)
          ((1 ldv) (1 *)))
        ldv work ldwork)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j k) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref c-%data%

```

```

((f2cl-lib:int-add
  (f2cl-lib:int-sub m k)
  j)
 i)
((1 ldc) (1 *))
c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    ((f2cl-lib:int-add
      (f2cl-lib:int-sub m k)
      j)
      i)
      ((1 ldc) (1 *))
      c-%offset%)
    (f2cl-lib:fref work-%data%
      (i j)
      ((1 ldwork) (1 *))
      work-%offset%))))))
(char-equal side #\R)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j k) nil)
  (tagbody
    (dcopy m
      (f2cl-lib:array-slice c
        double-float
        (1
          (f2cl-lib:int-add
            (f2cl-lib:int-sub n k)
            j))
          ((1 ldc) (1 *)))
        1
        (f2cl-lib:array-slice work
          double-float
          (1 j)
          ((1 ldwork) (1 *)))
        1)))
(dtrmm "Right" "Upper" "No transpose" "Unit" m k one
  (f2cl-lib:array-slice v
    double-float
    ((+ n (f2cl-lib:int-sub k) 1) 1)
    ((1 ldv) (1 *)))
  ldv work ldwork)
(cond
  ((> n k)
    (dgemm "No transpose" "No transpose" m k
      (f2cl-lib:int-sub n k) one c ldc v ldv one work ldwork)))
(dtrmm "Right" "Lower" trans "Non-unit" m k one t$ ldt work
  ldwork)
(cond
  ((> n k)

```



```

      (dgemm "No transpose" "Transpose" m (f2cl-lib:int-sub n k)
        k (- one) work ldwork v ldv one c ldc)))
(dtrmm "Right" "Upper" "Transpose" "Unit" m k one
  (f2cl-lib:array-slice v
    double-float
    ((+ n (f2cl-lib:int-sub k) 1) 1)
    ((1 ldv) (1 *)))
  ldv work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (i
        (f2cl-lib:int-add
          (f2cl-lib:int-sub n k)
          j))
        ((1 ldc) (1 *))
        c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (i
            (f2cl-lib:int-add
              (f2cl-lib:int-sub n k)
              j))
            ((1 ldc) (1 *))
            c-%offset%)
        (f2cl-lib:fref work-%data%
          (i j)
          ((1 ldwork) (1 *))
          work-%offset%)))))))))
((char-equal storev #\R)
  (cond
    ((char-equal direct #\F)
      (cond
        ((char-equal side #\L)
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            (> j k) nil)
          (tagbody
            (dcopy n
              (f2cl-lib:array-slice c
                double-float
                (j 1)
                ((1 ldc) (1 *)))
              ldc
              (f2cl-lib:array-slice work
                double-float
                (1 j)

```

```

((1 ldwork) (1 *)))
1)))
(dtrmm "Right" "Upper" "Transpose" "Unit" n k one v ldv work
ldwork)
(cond
  (> m k)
  (dgemm "Transpose" "Transpose" n k (f2cl-lib:int-sub m k)
one
  (f2cl-lib:array-slice c
double-float
  ((+ k 1) 1)
  ((1 ldc) (1 *)))

ldc
  (f2cl-lib:array-slice v
double-float
  (1 (f2cl-lib:int-add k 1))
  ((1 ldv) (1 *)))

ldv one work ldwork)))
(dtrmm "Right" "Upper" transt "Non-unit" n k one t$ ldt work
ldwork)
(cond
  (> m k)
  (dgemm "Transpose" "Transpose" (f2cl-lib:int-sub m k) n k
(- one)
  (f2cl-lib:array-slice v
double-float
  (1 (f2cl-lib:int-add k 1))
  ((1 ldv) (1 *)))

ldv work ldwork one
  (f2cl-lib:array-slice c
double-float
  ((+ k 1) 1)
  ((1 ldc) (1 *)))

ldc)))
(dtrmm "Right" "Upper" "No transpose" "Unit" n k one v ldv
work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
      (j i)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j i)
          ((1 ldc) (1 *))

```

```

                                c-%offset%)
                                (f2c1-lib:fref work-%data%
                                (i j)
                                ((1 ldwork) (1 *)))
                                work-%offset%)))))))))
(char-equal side #\R)
(f2c1-lib:fdo (j 1 (f2c1-lib:int-add j 1))
              (> j k) nil)
  (tagbody
    (dcopy m
      (f2c1-lib:array-slice c
        double-float
        (1 j)
        ((1 ldc) (1 *)))
      1
      (f2c1-lib:array-slice work
        double-float
        (1 j)
        ((1 ldwork) (1 *)))
      1)))
  (dtrmm "Right" "Upper" "Transpose" "Unit" m k one v ldv work
    ldwork)
  (cond
    ((> n k)
     (dgemm "No transpose" "Transpose" m k
       (f2c1-lib:int-sub n k) one
       (f2c1-lib:array-slice c
         double-float
         (1 (f2c1-lib:int-add k 1))
         ((1 ldc) (1 *)))
       ldc
       (f2c1-lib:array-slice v
         double-float
         (1 (f2c1-lib:int-add k 1))
         ((1 ldv) (1 *)))
       ldv one work ldwork)))
  (dtrmm "Right" "Upper" trans "Non-unit" m k one t$ ldt work
    ldwork)
  (cond
    ((> n k)
     (dgemm "No transpose" "No transpose" m
       (f2c1-lib:int-sub n k) k (- one) work ldwork
       (f2c1-lib:array-slice v
         double-float
         (1 (f2c1-lib:int-add k 1))
         ((1 ldv) (1 *)))
       ldv one
       (f2c1-lib:array-slice c
         double-float
         (1 (f2c1-lib:int-add k 1))

```

```

                                ((1 ldc) (1 *)))
      ldc)))
    (dtrmm "Right" "Upper" "No transpose" "Unit" m k one v ldv
     work ldwork)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j k) nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i m) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
                           (i j)
                           ((1 ldc) (1 *)))
              c-%offset%)
          (-
            (f2cl-lib:fref c-%data%
                          (i j)
                          ((1 ldc) (1 *)))
              c-%offset%)
            (f2cl-lib:fref work-%data%
                          (i j)
                          ((1 ldwork) (1 *)))
              work-%offset%)))))))))
  (t
   (cond
    ((char-equal side #\L)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
       (> j k) nil)
     (tagbody
       (dcopy n
        (f2cl-lib:array-slice c
                              double-float
                              ((+ m (f2cl-lib:int-sub k) j) 1)
                              ((1 ldc) (1 *)))

        ldc
        (f2cl-lib:array-slice work
                              double-float
                              (1 j)
                              ((1 ldwork) (1 *)))

        1)))
    (dtrmm "Right" "Lower" "Transpose" "Unit" n k one
     (f2cl-lib:array-slice v
                          double-float
                          (1
                           (f2cl-lib:int-add
                            (f2cl-lib:int-sub m k)
                            1))
                          ((1 ldv) (1 *)))

     ldv work ldwork)
    (cond

```

```

(> m k)
  (dgemm "Transpose" "Transpose" n k (f2cl-lib:int-sub m k)
    one c ldc v ldv one work ldwork)))
(dtrmm "Right" "Lower" transt "Non-unit" n k one t$ ldt work
  ldwork)
(cond
  (> m k)
    (dgemm "Transpose" "Transpose" (f2cl-lib:int-sub m k) n k
      (- one) v ldv work ldwork one c ldc)))
(dtrmm "Right" "Lower" "No transpose" "Unit" n k one
  (f2cl-lib:array-slice v
    double-float
    (1
      (f2cl-lib:int-add
        (f2cl-lib:int-sub m k)
        1))
    ((1 ldv) (1 *))))
  ldv work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data%
          ((f2cl-lib:int-add
            (f2cl-lib:int-sub m k)
            j)
            i)
          ((1 ldc) (1 *)))
          c-%offset%)
          (-
            (f2cl-lib:fref c-%data%
              ((f2cl-lib:int-add
                (f2cl-lib:int-sub m k)
                j)
                i)
              ((1 ldc) (1 *)))
              c-%offset%)
            (f2cl-lib:fref work-%data%
              (i j)
              ((1 ldwork) (1 *)))
              work-%offset%))))))
((char-equal side #\R)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j k) nil)
    (tagbody
      (dcopy m
        (f2cl-lib:array-slice c
          double-float

```

```

                                (1
                                (f2cl-lib:int-add
                                (f2cl-lib:int-sub n k)
                                j))
                                ((1 ldc) (1 *)))
1
(f2cl-lib:array-slice work
                        double-float
                        (1 j)
                        ((1 ldwork) (1 *)))
1)))
(dtrmm "Right" "Lower" "Transpose" "Unit" m k one
(f2cl-lib:array-slice v
                        double-float
                        (1
                        (f2cl-lib:int-add
                        (f2cl-lib:int-sub n k)
                        1))
                        ((1 ldv) (1 *)))
ldv work ldwork)
(cond
  (> n k)
  (dgemm "No transpose" "Transpose" m k
    (f2cl-lib:int-sub n k) one c ldc v ldv one work ldwork)))
(dtrmm "Right" "Lower" trans "Non-unit" m k one t$ ldt work
ldwork)
(cond
  (> n k)
  (dgemm "No transpose" "No transpose" m
    (f2cl-lib:int-sub n k) k (- one) work ldwork v ldv one c
    ldc)))
(dtrmm "Right" "Lower" "No transpose" "Unit" m k one
(f2cl-lib:array-slice v
                        double-float
                        (1
                        (f2cl-lib:int-add
                        (f2cl-lib:int-sub n k)
                        1))
                        ((1 ldv) (1 *)))
ldv work ldwork)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (i
                        (f2cl-lib:int-add
                        (f2cl-lib:int-sub n k)

```

dlarfg LAPACK

```
)set break resume
)sys rm -f dlarfg.output
)spool dlarfg.output
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dlarfg.help —

```
=====
dlarfg examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLARFG - a real elementary reflector H of order n, such that $H * \begin{pmatrix} \alpha \\ \text{alpha} \end{pmatrix} = \begin{pmatrix} \beta \\ \text{beta} \end{pmatrix}$, $H' * H = I$

SYNOPSIS

```
SUBROUTINE DLARFG( N, ALPHA, X, INCX, TAU )
```

```
      INTEGER      INCX, N
```

```
      DOUBLE      PRECISION ALPHA, TAU
```

```
      DOUBLE      PRECISION X( * )
```

Purpose

```
=====
```

DLARFG generates a real elementary reflector H of order n, such that

$$H * \begin{pmatrix} \alpha \\ \text{alpha} \\ \text{x} \end{pmatrix} = \begin{pmatrix} \beta \\ \text{beta} \\ 0 \end{pmatrix}, \quad H' * H = I.$$

where alpha and beta are scalars, and x is an (n-1)-element real vector. H is represented in the form

$$H = I - \tau * \begin{pmatrix} 1 \\ \text{v} \end{pmatrix} * \begin{pmatrix} 1 & \text{v}' \end{pmatrix},$$

where tau is a real scalar and v is a real (n-1)-element vector.

If the elements of x are all zero, then tau = 0 and H is taken to be the unit matrix.

Otherwise $1 \leq \tau \leq 2$.

Arguments

=====

N (input) INTEGER
The order of the elementary reflector.

ALPHA (input/output) DOUBLE PRECISION
On entry, the value alpha.
On exit, it is overwritten with the value beta.

X (input/output) DOUBLE PRECISION array, dimension
(1+(N-2)*abs(INCX))
On entry, the vector x.
On exit, it is overwritten with the vector v.

INCX (input) INTEGER
The increment between elements of X. INCX > 0.

TAU (output) DOUBLE PRECISION
The value tau.

— dlarfg.f —

```

SUBROUTINE DLARFG( N, ALPHA, X, INCX, TAU )
*
* -- LAPACK auxiliary routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
* Courant Institute, Argonne National Lab, and Rice University
* September 30, 1994
*
* .. Scalar Arguments ..
INTEGER          INCX, N
DOUBLE PRECISION ALPHA, TAU
*
* ..
* .. Array Arguments ..
DOUBLE PRECISION X( * )
*
* ..
*
* =====
*
* .. Parameters ..
DOUBLE PRECISION ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
* ..

```

```

*      .. Local Scalars ..
      INTEGER          J, KNT
      DOUBLE PRECISION BETA, RSAFMN, SAFMIN, XNORM
*
*      ..
*      .. External Functions ..
      DOUBLE PRECISION DLAMCH, DLAPY2, DNRM2
      EXTERNAL          DLAMCH, DLAPY2, DNRM2
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, SIGN
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DSCAL
*
*      ..
*      .. Executable Statements ..
*
      IF( N.LE.1 ) THEN
        TAU = ZERO
        RETURN
      END IF
*
      XNORM = DNRM2( N-1, X, INCX )
*
      IF( XNORM.EQ.ZERO ) THEN
*
*         H = I
*
*         TAU = ZERO
      ELSE
*
*         general case
*
      BETA = -SIGN( DLAPY2( ALPHA, XNORM ), ALPHA )
      SAFMIN = DLAMCH( 'S' ) / DLAMCH( 'E' )
      IF( ABS( BETA ).LT.SAFMIN ) THEN
*
*         XNORM, BETA may be inaccurate; scale X and recompute them
*
      RSAFMN = ONE / SAFMIN
      KNT = 0
10      CONTINUE
      KNT = KNT + 1
      CALL DSCAL( N-1, RSAFMN, X, INCX )
      BETA = BETA*RSAFMN
      ALPHA = ALPHA*RSAFMN
      IF( ABS( BETA ).LT.SAFMIN )
        $      GO TO 10
*
*         New BETA is at most 1, at least SAFMIN
*

```

```

      XNORM = DNRM2( N-1, X, INCX )
      BETA = -SIGN( DLAPY2( ALPHA, XNORM ), ALPHA )
      TAU = ( BETA-ALPHA ) / BETA
      CALL DSCAL( N-1, ONE / ( ALPHA-BETA ), X, INCX )
*
*      If ALPHA is subnormal, it may lose relative accuracy
*
      ALPHA = BETA
      DO 20 J = 1, KNT
        ALPHA = ALPHA*SAFMIN
20    CONTINUE
      ELSE
        TAU = ( BETA-ALPHA ) / BETA
        CALL DSCAL( N-1, ONE / ( ALPHA-BETA ), X, INCX )
        ALPHA = BETA
      END IF
    END IF
*
  RETURN
*
*  End of DLARFG
*
  END

```

— LAPACK dlarfg —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dlarfg (n alpha x incx tau)
    (declare (type (simple-array double-float (*)) x)
              (type (double-float) tau alpha)
              (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
      ((x double-float x-%data% x-%offset%))
      (prog ((beta 0.0) (rsafmn 0.0) (safmin 0.0) (xnrm 0.0) (j 0) (knt 0))
        (declare (type (double-float) beta rsafmn safmin xnrm)
                  (type fixnum j knt))
        (cond
          ((<= n 1)
           (setf tau zero)
           (go end_label)))
        (setf xnrm (dnrm2 (f2cl-lib:int-sub n 1) x incx))
        (cond
          ((= xnrm zero)
           (setf tau zero))

```

```

(t
  (setf beta (- (f2cl-lib:sign (dlapy2 alpha xnorm) alpha)))
  (setf safmin (/ (dlamch "S") (dlamch "E")))
  (cond
    ((< (abs beta) safmin)
      (tagbody
        (setf rsafmn (/ one safmin))
        (setf knt 0)
label10
        (setf knt (f2cl-lib:int-add knt 1))
        (dscal (f2cl-lib:int-sub n 1) rsafmn x incx)
        (setf beta (* beta rsafmn))
        (setf alpha (* alpha rsafmn))
        (if (< (abs beta) safmin) (go label10))
        (setf xnorm (dnrm2 (f2cl-lib:int-sub n 1) x incx))
        (setf beta (- (f2cl-lib:sign (dlapy2 alpha xnorm) alpha)))
        (setf tau (/ (- beta alpha) beta))
        (dscal (f2cl-lib:int-sub n 1) (/ one (- alpha beta)) x incx)
        (setf alpha beta)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j knt) nil)
          (tagbody (setf alpha (* alpha safmin)) label20))))
  (t
    (setf tau (/ (- beta alpha) beta))
    (dscal (f2cl-lib:int-sub n 1) (/ one (- alpha beta)) x incx)
    (setf alpha beta))))
end_label
(return (values nil alpha nil nil tau))))

```

dlarf LAPACK

— dlarf.input —

```

)set break resume
)sys rm -f dlarf.output
)spool dlarf.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlarf.help —

```
=====
dlarf examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLARF - a real elementary reflector H to a real m by n matrix C, from either the left or the right

SYNOPSIS

```
SUBROUTINE DLARF( SIDE, M, N, V, INCV, TAU, C, LDC, WORK )
```

```

      CHARACTER      SIDE
      INTEGER        INCV, LDC, M, N
      DOUBLE         PRECISION TAU
      DOUBLE         PRECISION C( LDC, * ), V( * ), WORK( * )
```

Purpose

```
=====
```

DLARF applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

$$H = I - \tau * v * v'$$

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix.

Arguments

```
=====
```

```

SIDE      (input) CHARACTER*1
          = 'L': form  H * C
          = 'R': form  C * H

M          (input) INTEGER
          The number of rows of the matrix C.

N          (input) INTEGER
          The number of columns of the matrix C.
```

V (input) DOUBLE PRECISION array, dimension
 $(1 + (M-1)*abs(INCV))$ if SIDE = 'L'
 or $(1 + (N-1)*abs(INCV))$ if SIDE = 'R'
 The vector v in the representation of H. V is not used if
 TAU = 0.

INCV (input) INTEGER
 The increment between elements of v. INCV \neq 0.

TAU (input) DOUBLE PRECISION
 The value tau in the representation of H.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the m by n matrix C.
 On exit, C is overwritten by the matrix $H * C$ if SIDE = 'L',
 or $C * H$ if SIDE = 'R'.

LDC (input) INTEGER
 The leading dimension of the array C. LDC $\geq \max(1,M)$.

WORK (workspace) DOUBLE PRECISION array, dimension
 (N) if SIDE = 'L'
 or (M) if SIDE = 'R'

— dlarf.f —

```

SUBROUTINE DLARF( SIDE, M, N, V, INCV, TAU, C, LDC, WORK )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    February 29, 1992
*
*    .. Scalar Arguments ..
*    CHARACTER          SIDE
*    INTEGER            INCV, LDC, M, N
*    DOUBLE PRECISION   TAU
*
*    ..
*    .. Array Arguments ..
*    DOUBLE PRECISION   C( LDC, * ), V( * ), WORK( * )
*
*    ..
*
*    =====
*
*    .. Parameters ..

```

```

      DOUBLE PRECISION  ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      ..
*      .. External Subroutines ..
      EXTERNAL           DGEMV, DGER
*
*      ..
*      .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*
*      ..
*      .. Executable Statements ..
*
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*         Form  H * C
*
*         IF( TAU.NE.ZERO ) THEN
*
*            w := C' * v
*
*            CALL DGEMV( 'Transpose', M, N, ONE, C, LDC, V, INCV, ZERO,
$              WORK, 1 )
*
*            C := C - v * w'
*
*            CALL DGER( M, N, -TAU, V, INCV, WORK, 1, C, LDC )
          END IF
        ELSE
*
*         Form  C * H
*
*         IF( TAU.NE.ZERO ) THEN
*
*            w := C * v
*
*            CALL DGEMV( 'No transpose', M, N, ONE, C, LDC, V, INCV,
$              ZERO, WORK, 1 )
*
*            C := C - w * v'
*
*            CALL DGER( M, N, -TAU, WORK, 1, V, INCV, C, LDC )
          END IF
        END IF
      RETURN
*
*      End of DLARF
*
      END

```

— LAPACK dlarf —

```
(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlarf (side m n v incv tau c ldc work)
    (declare (type (double-float) tau)
             (type (simple-array double-float (*)) work c v)
             (type fixnum ldc incv n m)
             (type character side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (v double-float v-%data% v-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ()
        (declare)
        (cond
          ((char-equal side #\L)
           (cond
            ((/= tau zero)
             (dgemv "Transpose" m n one c ldc v incv zero work 1)
             (dger m n (- tau) v incv work 1 c ldc))))
          (t
           (cond
            ((/= tau zero)
             (dgemv "No transpose" m n one c ldc v incv zero work 1)
             (dger m n (- tau) work 1 v incv c ldc))))))
      (return (values nil nil nil nil nil nil nil nil))))))
```

dlarft LAPACK

— dlarft.input —

```
)set break resume
)sys rm -f dlarft.output
)spool dlarft.output
)set message test on
)set message auto off
)clear all
```



```
)spool
)lisp (bye)
```

— dlarft.help —

```
=====
dlarft examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLARFT - the triangular factor T of a real block reflector H of order n, which is defined as a product of k elementary reflectors

SYNOPSIS

```
SUBROUTINE DLARFT( DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT )
```

```
      CHARACTER      DIRECT, STOREV
```

```
      INTEGER        K, LDT, LDV, N
```

```
      DOUBLE         PRECISION T( LDT, * ), TAU( * ), V( LDV, * )
```

Purpose

```
=====
```

DLARFT forms the triangular factor T of a real block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

$$H = I - V * T * V'$$

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

$$H = I - V' * T * V$$

Arguments

DIRECT (input) CHARACTER*1
Specifies the order in which the elementary reflectors are multiplied to form the block reflector:
= 'F': $H = H(1) H(2) \dots H(k)$ (Forward)
= 'B': $H = H(k) \dots H(2) H(1)$ (Backward)

STOREV (input) CHARACTER*1
Specifies how the vectors which define the elementary reflectors are stored (see also Further Details):
= 'C': columnwise
= 'R': rowwise

N (input) INTEGER
The order of the block reflector H . $N \geq 0$.

K (input) INTEGER
The order of the triangular factor T (= the number of elementary reflectors). $K \geq 1$.

V (input/output) DOUBLE PRECISION array, dimension
(LDV,K) if STOREV = 'C'
(LDV,N) if STOREV = 'R'
The matrix V . See further details.

LDV (input) INTEGER
The leading dimension of the array V .
If STOREV = 'C', $LDV \geq \max(1,N)$; if STOREV = 'R', $LDV \geq K$.

TAU (input) DOUBLE PRECISION array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector $H(i)$.

T (output) DOUBLE PRECISION array, dimension (LDT,K)
The k by k triangular factor T of the block reflector.
If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is lower triangular. The rest of the array is not used.

LDT (input) INTEGER
The leading dimension of the array T . $LDT \geq K$.

=====

The shape of the matrix V and the storage of the vectors which define the H(i) is best illustrated by the following example with $n = 5$ and $k = 3$. The elements equal to 1 are not stored; the corresponding array elements are modified but restored on exit. The rest of the array is not used.

DIRECT = 'F' and STOREV = 'C':

$$V = \begin{pmatrix} 1 & & \\ v1 & 1 & \\ v1 & v2 & 1 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \end{pmatrix}$$

DIRECT = 'F' and STOREV = 'R':

$$V = \begin{pmatrix} 1 & v1 & v1 & v1 & v1 \\ & 1 & v2 & v2 & v2 \\ & & 1 & v3 & v3 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'C':

$$V = \begin{pmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ 1 & v2 & v3 \\ & 1 & v3 \\ & & 1 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'R':

$$V = \begin{pmatrix} v1 & v1 & 1 & & \\ v2 & v2 & v2 & 1 & \\ v3 & v3 & v3 & v3 & 1 \end{pmatrix}$$

— dlarft.f —

```

SUBROUTINE DLARFT( DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
CHARACTER      DIRECT, STOREV
INTEGER        K, LDT, LDV, N
*
*  ..
*
*  .. Array Arguments ..
DOUBLE PRECISION  T( LDT, * ), TAU( * ), V( LDV, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION  ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
INTEGER          I, J
DOUBLE PRECISION  VII
*
*  ..
*
*  .. External Subroutines ..
EXTERNAL         DGEMV, DTRMV

```

```

*      ..
*      .. External Functions ..
*      LOGICAL          LSAME
*      EXTERNAL          LSAME
*      ..
*      .. Executable Statements ..
*
*      Quick return if possible
*
*      IF( N.EQ.0 )
*      $   RETURN
*
*      IF( LSAME( DIRECT, 'F' ) ) THEN
*          DO 20 I = 1, K
*              IF( TAU( I ).EQ.ZERO ) THEN
*
*                  H(i) = I
*
*                  DO 10 J = 1, I
*                      T( J, I ) = ZERO
*
*                  10 CONTINUE
*              ELSE
*
*                  general case
*
*                  VII = V( I, I )
*                  V( I, I ) = ONE
*                  IF( LSAME( STOREV, 'C' ) ) THEN
*
*                      T(1:i-1,i) := - tau(i) * V(i:n,1:i-1)' * V(i:n,i)
*
*                      CALL DGEMV( 'Transpose', N-I+1, I-1, -TAU( I ),
*      $                          V( I, 1 ), LDV, V( I, I ), 1, ZERO,
*      $                          T( 1, I ), 1 )
*
*                      ELSE
*
*                          T(1:i-1,i) := - tau(i) * V(1:i-1,i:n) * V(i,i:n)'
*
*                          CALL DGEMV( 'No transpose', I-1, N-I+1, -TAU( I ),
*      $                          V( 1, I ), LDV, V( I, I ), LDV, ZERO,
*      $                          T( 1, I ), 1 )
*
*                      END IF
*                      V( I, I ) = VII
*
*                      T(1:i-1,i) := T(1:i-1,1:i-1) * T(1:i-1,i)
*
*                      CALL DTRMV( 'Upper', 'No transpose', 'Non-unit', I-1, T,
*      $                          LDT, T( 1, I ), 1 )
*                      T( I, I ) = TAU( I )
*                  END IF

```

```

20    CONTINUE
    ELSE
        DO 40 I = K, 1, -1
            IF( TAU( I ).EQ.ZERO ) THEN
*
*              H(i) = I
*
                DO 30 J = I, K
                    T( J, I ) = ZERO
30            CONTINUE
            ELSE
*
*              general case
*
                IF( I.LT.K ) THEN
                    IF( LSAME( STOREV, 'C' ) ) THEN
                        VII = V( N-K+I, I )
                        V( N-K+I, I ) = ONE
*
*              T(i+1:k,i) :=
*              - tau(i) * V(1:n-k+i,i+1:k)' * V(1:n-k+i,i)
*
                        CALL DGEMV( 'Transpose', N-K+I, K-I, -TAU( I ),
$                          V( 1, I+1 ), LDV, V( 1, I ), 1, ZERO,
$                          T( I+1, I ), 1 )
                        V( N-K+I, I ) = VII
                    ELSE
                        VII = V( I, N-K+I )
                        V( I, N-K+I ) = ONE
*
*              T(i+1:k,i) :=
*              - tau(i) * V(i+1:k,1:n-k+i) * V(i,1:n-k+i)'
*
                        CALL DGEMV( 'No transpose', K-I, N-K+I, -TAU( I ),
$                          V( I+1, 1 ), LDV, V( I, 1 ), LDV, ZERO,
$                          T( I+1, I ), 1 )
                        V( I, N-K+I ) = VII
                    END IF
*
*              T(i+1:k,i) := T(i+1:k,i+1:k) * T(i+1:k,i)
*
                        CALL DTRMV( 'Lower', 'No transpose', 'Non-unit', K-I,
$                          T( I+1, I+1 ), LDT, T( I+1, I ), 1 )
                    END IF
                    T( I, I ) = TAU( I )
                END IF
            CONTINUE
40        CONTINUE
    END IF
    RETURN
*

```

```

*      End of DLARFT
*
      END

```

— LAPACK dlarft —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dlarft (direct storev n k v ldv tau t$ ldt)
    (declare (type (simple-array double-float (*)) t$ tau v)
              (type fixnum ldt ldv k n)
              (type character storev direct))
    (f2cl-lib:with-multi-array-data
      ((direct character direct-%data% direct-%offset%)
       (storev character storev-%data% storev-%offset%)
       (v double-float v-%data% v-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (t$ double-float t$-%data% t$-%offset%))
      (prog ((vii 0.0) (i 0) (j 0))
        (declare (type (double-float) vii) (type fixnum i j))
        (if (= n 0) (go end_label))
        (cond
          ((char-equal direct #\F)
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                         (> i k) nil)
           (tagbody
            (cond
              ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
               (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                             (> j i) nil)
               (tagbody
                (setf (f2cl-lib:fref t$-%data%
                                     (j i)
                                     ((1 ldt) (1 *))
                                     t$-%offset%)
                      zero))))
              (t
               (setf vii
                     (f2cl-lib:fref v-%data%
                                     (i i)
                                     ((1 ldv) (1 *))
                                     v-%offset%))
               (setf (f2cl-lib:fref v-%data%
                                     (i i)
                                     ((1 ldv) (1 *))

```

```

                                v-%offset%)
                                one)
(cond
  ((char-equal storev #\C)
   (dgemv "Transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
    (f2cl-lib:int-sub i 1)
    (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
    (f2cl-lib:array-slice v
                           double-float
                           (i 1)
                           ((1 ldv) (1 *)))

    ldv
    (f2cl-lib:array-slice v
                           double-float
                           (i i)
                           ((1 ldv) (1 *)))

    1 zero
    (f2cl-lib:array-slice t$
                           double-float
                           (1 i)
                           ((1 ldt) (1 *)))

    1))
  (t
   (dgemv "No transpose" (f2cl-lib:int-sub i 1)
    (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
    (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
    (f2cl-lib:array-slice v
                           double-float
                           (1 i)
                           ((1 ldv) (1 *)))

    ldv
    (f2cl-lib:array-slice v
                           double-float
                           (i i)
                           ((1 ldv) (1 *)))

    ldv zero
    (f2cl-lib:array-slice t$
                           double-float
                           (1 i)
                           ((1 ldt) (1 *)))

    1)))
(setf (f2cl-lib:fref v-%data%
                     (i i)
                     ((1 ldv) (1 *))
                     v-%offset%)
      vii)
(dtrmv "Upper" "No transpose" "Non-unit"
 (f2cl-lib:int-sub i 1) t$ ldt
 (f2cl-lib:array-slice t$ double-float (1 i) ((1 ldt) (1 *))))

```

```

1)
(setf (f2cl-lib:fref t$-%data%
                    (i i)
                    ((1 ldt) (1 *))
                    t$-%offset%)
      (f2cl-lib:fref tau-%data%
                    (i)
                    ((1 *))
                    tau-%offset%))))))
(t
 (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
               ((> i 1) nil)
  (tagbody
   (cond
    ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
     (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
                   ((> j k) nil)
      (tagbody
       (setf (f2cl-lib:fref t$-%data%
                           (j i)
                           ((1 ldt) (1 *))
                           t$-%offset%)
              zero))))
    (t
     (cond
      ((< i k)
       (cond
        ((char-equal storev #\C)
         (setf vii
                (f2cl-lib:fref v-%data%
                              ((f2cl-lib:int-add
                               (f2cl-lib:int-sub n k)
                               i)
                               i)
                              ((1 ldv) (1 *))
                              v-%offset%))
              (setf (f2cl-lib:fref v-%data%
                              ((f2cl-lib:int-add
                               (f2cl-lib:int-sub n k)
                               i)
                               i)
                              ((1 ldv) (1 *))
                              v-%offset%)
                    one)
              (dgemv "Transpose"
                     (f2cl-lib:int-add (f2cl-lib:int-sub n k) i)
                     (f2cl-lib:int-sub k i)
                     (-
                      (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                      (f2cl-lib:array-slice v

```



```

                                double-float
                                (1 (f2cl-lib:int-add i 1))
                                ((1 ldv) (1 *)))

ldv
(f2cl-lib:array-slice v
  double-float
  (1 i)
  ((1 ldv) (1 *)))

1 zero
(f2cl-lib:array-slice t$
  double-float
  ((+ i 1) i)
  ((1 ldt) (1 *)))

1)
(setf (f2cl-lib:fref v-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub n k)
    i)
    i)
  ((1 ldv) (1 *))
  v-%offset%)
  vii))

(t
  (setf vii
    (f2cl-lib:fref v-%data%
      (i
        (f2cl-lib:int-add
          (f2cl-lib:int-sub n k)
          i))
        ((1 ldv) (1 *))
        v-%offset%))
    (setf (f2cl-lib:fref v-%data%
      (i
        (f2cl-lib:int-add
          (f2cl-lib:int-sub n k)
          i))
        ((1 ldv) (1 *))
        v-%offset%)
      one)
    (dgemv "No transpose" (f2cl-lib:int-sub k i)
      (f2cl-lib:int-add (f2cl-lib:int-sub n k) i)
      (-
        (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
      (f2cl-lib:array-slice v
        double-float
        ((+ i 1) 1)
        ((1 ldv) (1 *)))

ldv
(f2cl-lib:array-slice v
  double-float

```

```

                                (i 1)
                                ((1 ldv) (1 *)))
ldv zero
(f2cl-lib:array-slice t$
  double-float
  ((+ i 1) i)
  ((1 ldt) (1 *)))
1)
(setf (f2cl-lib:fref v-%data%
  (i
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n k)
      i))
    ((1 ldv) (1 *))
    v-%offset%
    vii)))
(dtrmv "Lower" "No transpose" "Non-unit"
  (f2cl-lib:int-sub k i)
  (f2cl-lib:array-slice t$
    double-float
    ((+ i 1) (f2cl-lib:int-add i 1))
    ((1 ldt) (1 *)))
ldt
(f2cl-lib:array-slice t$
  double-float
  ((+ i 1) i)
  ((1 ldt) (1 *)))
1)))
(setf (f2cl-lib:fref t$-%data%
  (i i)
  ((1 ldt) (1 *))
  t$-%offset%
  (f2cl-lib:fref tau-%data%
    (i)
    ((1 *))
    tau-%offset%))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))

```

dlarfx LAPACK

— dlarfx.input —

```
)set break resume
```

```

)sys rm -f dlarfx.output
)spool dlarfx.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlarfx.help —

```

=====
dlarfx examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLARFX - a real elementary reflector H to a real m by n matrix C, from either the left or the right

SYNOPSIS

SUBROUTINE DLARFX(SIDE, M, N, V, TAU, C, LDC, WORK)

| | |
|-----------|--|
| CHARACTER | SIDE |
| INTEGER | LDC, M, N |
| DOUBLE | PRECISION TAU |
| DOUBLE | PRECISION C(LDC, *), V(*), WORK(*) |

Purpose

```

=====

```

DLARFX applies a real elementary reflector H to a real m by n matrix C, from either the left or the right. H is represented in the form

$$H = I - \tau * v * v'$$

where tau is a real scalar and v is a real vector.

If tau = 0, then H is taken to be the unit matrix

This version uses inline code if H has order < 11 .

Arguments

=====

SIDE (input) CHARACTER*1
 = 'L': form $H * C$
 = 'R': form $C * H$

M (input) INTEGER
 The number of rows of the matrix C.

N (input) INTEGER
 The number of columns of the matrix C.

V (input) DOUBLE PRECISION array, dimension (M) if SIDE = 'L'
 or (N) if SIDE = 'R'
 The vector v in the representation of H.

TAU (input) DOUBLE PRECISION
 The value tau in the representation of H.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the m by n matrix C.
 On exit, C is overwritten by the matrix $H * C$ if SIDE = 'L',
 or $C * H$ if SIDE = 'R'.

LDC (input) INTEGER
 The leading dimension of the array C. LDA \geq (1,M).

WORK (workspace) DOUBLE PRECISION array, dimension
 (N) if SIDE = 'L'
 or (M) if SIDE = 'R'
 WORK is not referenced if H has order < 11 .

— dlarfx.f —

```

SUBROUTINE DLARFX( SIDE, M, N, V, TAU, C, LDC, WORK )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    February 29, 1992
*
*    .. Scalar Arguments ..
CHARACTER          SIDE
```

```

      INTEGER          LDC, M, N
      DOUBLE PRECISION TAU
*
*   ..
*   .. Array Arguments ..
      DOUBLE PRECISION C( LDC, * ), V( * ), WORK( * )
*
*   ..
*
*   =====
*
*   .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*   ..
*   .. Local Scalars ..
      INTEGER            J
      DOUBLE PRECISION   SUM, T1, T10, T2, T3, T4, T5, T6, T7, T8, T9,
$                       V1, V10, V2, V3, V4, V5, V6, V7, V8, V9
*
*   ..
*   .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*
*   ..
*   .. External Subroutines ..
      EXTERNAL           DGEMV, DGER
*
*   ..
*   .. Executable Statements ..
*
      IF( TAU.EQ.ZERO )
$      RETURN
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*       Form  $H * C$ , where H has order m.
*
*       GO TO ( 10, 30, 50, 70, 90, 110, 130, 150,
$           170, 190 )M
*
*       Code for general M
*
*        $w := C' * v$ 
*
      CALL DGEMV( 'Transpose', M, N, ONE, C, LDC, V, 1, ZERO, WORK,
$           1 )
*
*        $C := C - \tau * v * w'$ 
*
      CALL DGER( M, N, -TAU, V, 1, WORK, 1, C, LDC )
      GO TO 410
10    CONTINUE
*
*       Special code for 1 x 1 Householder

```

```

*
      T1 = ONE - TAU*V( 1 )*V( 1 )
      DO 20 J = 1, N
        C( 1, J ) = T1*C( 1, J )
20    CONTINUE
      GO TO 410
30    CONTINUE
*
*      Special code for 2 x 2 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      DO 40 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
40    CONTINUE
      GO TO 410
50    CONTINUE
*
*      Special code for 3 x 3 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      DO 60 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
        C( 3, J ) = C( 3, J ) - SUM*T3
60    CONTINUE
      GO TO 410
70    CONTINUE
*
*      Special code for 4 x 4 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      DO 80 J = 1, N

```

```

      SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
$      V4*C( 4, J )
      C( 1, J ) = C( 1, J ) - SUM*T1
      C( 2, J ) = C( 2, J ) - SUM*T2
      C( 3, J ) = C( 3, J ) - SUM*T3
      C( 4, J ) = C( 4, J ) - SUM*T4
80  CONTINUE
      GO TO 410
90  CONTINUE
*
*      Special code for 5 x 5 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      DO 100 J = 1, N
$      SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
      V4*C( 4, J ) + V5*C( 5, J )
      C( 1, J ) = C( 1, J ) - SUM*T1
      C( 2, J ) = C( 2, J ) - SUM*T2
      C( 3, J ) = C( 3, J ) - SUM*T3
      C( 4, J ) = C( 4, J ) - SUM*T4
      C( 5, J ) = C( 5, J ) - SUM*T5
100 CONTINUE
      GO TO 410
110 CONTINUE
*
*      Special code for 6 x 6 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      DO 120 J = 1, N
      SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +

```

```

$          V4*C( 4, J ) + V5*C( 5, J ) + V6*C( 6, J )
      C( 1, J ) = C( 1, J ) - SUM*T1
      C( 2, J ) = C( 2, J ) - SUM*T2
      C( 3, J ) = C( 3, J ) - SUM*T3
      C( 4, J ) = C( 4, J ) - SUM*T4
      C( 5, J ) = C( 5, J ) - SUM*T5
      C( 6, J ) = C( 6, J ) - SUM*T6
120    CONTINUE
      GO TO 410
130    CONTINUE
*
*      Special code for 7 x 7 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7
      DO 140 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
$          V4*C( 4, J ) + V5*C( 5, J ) + V6*C( 6, J ) +
$          V7*C( 7, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
        C( 3, J ) = C( 3, J ) - SUM*T3
        C( 4, J ) = C( 4, J ) - SUM*T4
        C( 5, J ) = C( 5, J ) - SUM*T5
        C( 6, J ) = C( 6, J ) - SUM*T6
        C( 7, J ) = C( 7, J ) - SUM*T7
140    CONTINUE
      GO TO 410
150    CONTINUE
*
*      Special code for 8 x 8 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3

```



```

      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7
      V8 = V( 8 )
      T8 = TAU*V8
      DO 160 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
$         V4*C( 4, J ) + V5*C( 5, J ) + V6*C( 6, J ) +
$         V7*C( 7, J ) + V8*C( 8, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
        C( 3, J ) = C( 3, J ) - SUM*T3
        C( 4, J ) = C( 4, J ) - SUM*T4
        C( 5, J ) = C( 5, J ) - SUM*T5
        C( 6, J ) = C( 6, J ) - SUM*T6
        C( 7, J ) = C( 7, J ) - SUM*T7
        C( 8, J ) = C( 8, J ) - SUM*T8
160    CONTINUE
      GO TO 410
170    CONTINUE
*
*      Special code for 9 x 9 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7
      V8 = V( 8 )
      T8 = TAU*V8
      V9 = V( 9 )
      T9 = TAU*V9
      DO 180 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
$         V4*C( 4, J ) + V5*C( 5, J ) + V6*C( 6, J ) +
$         V7*C( 7, J ) + V8*C( 8, J ) + V9*C( 9, J )

```

```

      C( 1, J ) = C( 1, J ) - SUM*T1
      C( 2, J ) = C( 2, J ) - SUM*T2
      C( 3, J ) = C( 3, J ) - SUM*T3
      C( 4, J ) = C( 4, J ) - SUM*T4
      C( 5, J ) = C( 5, J ) - SUM*T5
      C( 6, J ) = C( 6, J ) - SUM*T6
      C( 7, J ) = C( 7, J ) - SUM*T7
      C( 8, J ) = C( 8, J ) - SUM*T8
      C( 9, J ) = C( 9, J ) - SUM*T9
180    CONTINUE
      GO TO 410
190    CONTINUE
*
*      Special code for 10 x 10 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7
      V8 = V( 8 )
      T8 = TAU*V8
      V9 = V( 9 )
      T9 = TAU*V9
      V10 = V( 10 )
      T10 = TAU*V10
      DO 200 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) + V3*C( 3, J ) +
$          V4*C( 4, J ) + V5*C( 5, J ) + V6*C( 6, J ) +
$          V7*C( 7, J ) + V8*C( 8, J ) + V9*C( 9, J ) +
$          V10*C( 10, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
        C( 3, J ) = C( 3, J ) - SUM*T3
        C( 4, J ) = C( 4, J ) - SUM*T4
        C( 5, J ) = C( 5, J ) - SUM*T5
        C( 6, J ) = C( 6, J ) - SUM*T6
        C( 7, J ) = C( 7, J ) - SUM*T7
        C( 8, J ) = C( 8, J ) - SUM*T8
        C( 9, J ) = C( 9, J ) - SUM*T9
        C( 10, J ) = C( 10, J ) - SUM*T10

```

```

200    CONTINUE
      GO TO 410
    ELSE
*
*      Form  $C * H$ , where  $H$  has order  $n$ .
*
      GO TO ( 210, 230, 250, 270, 290, 310, 330, 350,
$          370, 390 )N
*
*      Code for general  $N$ 
*
      w := C * v
*
      CALL DGEMV( 'No transpose', M, N, ONE, C, LDC, V, 1, ZERO,
$          WORK, 1 )
*
      C := C - tau * w * v'
*
      CALL DGER( M, N, -TAU, WORK, 1, V, 1, C, LDC )
      GO TO 410
210    CONTINUE
*
*      Special code for 1 x 1 Householder
*
      T1 = ONE - TAU*V( 1 )*V( 1 )
      DO 220 J = 1, M
        C( J, 1 ) = T1*C( J, 1 )
220    CONTINUE
      GO TO 410
230    CONTINUE
*
*      Special code for 2 x 2 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      DO 240 J = 1, M
        SUM = V1*C( J, 1 ) + V2*C( J, 2 )
        C( J, 1 ) = C( J, 1 ) - SUM*T1
        C( J, 2 ) = C( J, 2 ) - SUM*T2
240    CONTINUE
      GO TO 410
250    CONTINUE
*
*      Special code for 3 x 3 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )

```

```

T2 = TAU*V2
V3 = V( 3 )
T3 = TAU*V3
DO 260 J = 1, M
    SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 )
    C( J, 1 ) = C( J, 1 ) - SUM*T1
    C( J, 2 ) = C( J, 2 ) - SUM*T2
    C( J, 3 ) = C( J, 3 ) - SUM*T3
260  CONTINUE
    GO TO 410
270  CONTINUE
*
*      Special code for 4 x 4 Householder
*
V1 = V( 1 )
T1 = TAU*V1
V2 = V( 2 )
T2 = TAU*V2
V3 = V( 3 )
T3 = TAU*V3
V4 = V( 4 )
T4 = TAU*V4
DO 280 J = 1, M
    SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$      V4*C( J, 4 )
    C( J, 1 ) = C( J, 1 ) - SUM*T1
    C( J, 2 ) = C( J, 2 ) - SUM*T2
    C( J, 3 ) = C( J, 3 ) - SUM*T3
    C( J, 4 ) = C( J, 4 ) - SUM*T4
280  CONTINUE
    GO TO 410
290  CONTINUE
*
*      Special code for 5 x 5 Householder
*
V1 = V( 1 )
T1 = TAU*V1
V2 = V( 2 )
T2 = TAU*V2
V3 = V( 3 )
T3 = TAU*V3
V4 = V( 4 )
T4 = TAU*V4
V5 = V( 5 )
T5 = TAU*V5
DO 300 J = 1, M
    SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$      V4*C( J, 4 ) + V5*C( J, 5 )
    C( J, 1 ) = C( J, 1 ) - SUM*T1
    C( J, 2 ) = C( J, 2 ) - SUM*T2

```

```

          C( J, 3 ) = C( J, 3 ) - SUM*T3
          C( J, 4 ) = C( J, 4 ) - SUM*T4
          C( J, 5 ) = C( J, 5 ) - SUM*T5
300    CONTINUE
      GO TO 410
310    CONTINUE
*
*      Special code for 6 x 6 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      DO 320 J = 1, M
          SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$          V4*C( J, 4 ) + V5*C( J, 5 ) + V6*C( J, 6 )
          C( J, 1 ) = C( J, 1 ) - SUM*T1
          C( J, 2 ) = C( J, 2 ) - SUM*T2
          C( J, 3 ) = C( J, 3 ) - SUM*T3
          C( J, 4 ) = C( J, 4 ) - SUM*T4
          C( J, 5 ) = C( J, 5 ) - SUM*T5
          C( J, 6 ) = C( J, 6 ) - SUM*T6
320    CONTINUE
      GO TO 410
330    CONTINUE
*
*      Special code for 7 x 7 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7

```

```

DO 340 J = 1, M
    SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$       V4*C( J, 4 ) + V5*C( J, 5 ) + V6*C( J, 6 ) +
$       V7*C( J, 7 )
    C( J, 1 ) = C( J, 1 ) - SUM*T1
    C( J, 2 ) = C( J, 2 ) - SUM*T2
    C( J, 3 ) = C( J, 3 ) - SUM*T3
    C( J, 4 ) = C( J, 4 ) - SUM*T4
    C( J, 5 ) = C( J, 5 ) - SUM*T5
    C( J, 6 ) = C( J, 6 ) - SUM*T6
    C( J, 7 ) = C( J, 7 ) - SUM*T7
340  CONTINUE
    GO TO 410
350  CONTINUE
*
*      Special code for 8 x 8 Householder
*
    V1 = V( 1 )
    T1 = TAU*V1
    V2 = V( 2 )
    T2 = TAU*V2
    V3 = V( 3 )
    T3 = TAU*V3
    V4 = V( 4 )
    T4 = TAU*V4
    V5 = V( 5 )
    T5 = TAU*V5
    V6 = V( 6 )
    T6 = TAU*V6
    V7 = V( 7 )
    T7 = TAU*V7
    V8 = V( 8 )
    T8 = TAU*V8
DO 360 J = 1, M
    SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$       V4*C( J, 4 ) + V5*C( J, 5 ) + V6*C( J, 6 ) +
$       V7*C( J, 7 ) + V8*C( J, 8 )
    C( J, 1 ) = C( J, 1 ) - SUM*T1
    C( J, 2 ) = C( J, 2 ) - SUM*T2
    C( J, 3 ) = C( J, 3 ) - SUM*T3
    C( J, 4 ) = C( J, 4 ) - SUM*T4
    C( J, 5 ) = C( J, 5 ) - SUM*T5
    C( J, 6 ) = C( J, 6 ) - SUM*T6
    C( J, 7 ) = C( J, 7 ) - SUM*T7
    C( J, 8 ) = C( J, 8 ) - SUM*T8
360  CONTINUE
    GO TO 410
370  CONTINUE
*
*      Special code for 9 x 9 Householder

```

```

*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6
      V7 = V( 7 )
      T7 = TAU*V7
      V8 = V( 8 )
      T8 = TAU*V8
      V9 = V( 9 )
      T9 = TAU*V9
      DO 380 J = 1, M
          SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$          V4*C( J, 4 ) + V5*C( J, 5 ) + V6*C( J, 6 ) +
$          V7*C( J, 7 ) + V8*C( J, 8 ) + V9*C( J, 9 )
          C( J, 1 ) = C( J, 1 ) - SUM*T1
          C( J, 2 ) = C( J, 2 ) - SUM*T2
          C( J, 3 ) = C( J, 3 ) - SUM*T3
          C( J, 4 ) = C( J, 4 ) - SUM*T4
          C( J, 5 ) = C( J, 5 ) - SUM*T5
          C( J, 6 ) = C( J, 6 ) - SUM*T6
          C( J, 7 ) = C( J, 7 ) - SUM*T7
          C( J, 8 ) = C( J, 8 ) - SUM*T8
          C( J, 9 ) = C( J, 9 ) - SUM*T9
380      CONTINUE
          GO TO 410
390      CONTINUE
*
*      Special code for 10 x 10 Householder
*
      V1 = V( 1 )
      T1 = TAU*V1
      V2 = V( 2 )
      T2 = TAU*V2
      V3 = V( 3 )
      T3 = TAU*V3
      V4 = V( 4 )
      T4 = TAU*V4
      V5 = V( 5 )
      T5 = TAU*V5
      V6 = V( 6 )
      T6 = TAU*V6

```

```

      V7 = V( 7 )
      T7 = TAU*V7
      V8 = V( 8 )
      T8 = TAU*V8
      V9 = V( 9 )
      T9 = TAU*V9
      V10 = V( 10 )
      T10 = TAU*V10
      DO 400 J = 1, M
          SUM = V1*C( J, 1 ) + V2*C( J, 2 ) + V3*C( J, 3 ) +
$          V4*C( J, 4 ) + V5*C( J, 5 ) + V6*C( J, 6 ) +
$          V7*C( J, 7 ) + V8*C( J, 8 ) + V9*C( J, 9 ) +
$          V10*C( J, 10 )
          C( J, 1 ) = C( J, 1 ) - SUM*T1
          C( J, 2 ) = C( J, 2 ) - SUM*T2
          C( J, 3 ) = C( J, 3 ) - SUM*T3
          C( J, 4 ) = C( J, 4 ) - SUM*T4
          C( J, 5 ) = C( J, 5 ) - SUM*T5
          C( J, 6 ) = C( J, 6 ) - SUM*T6
          C( J, 7 ) = C( J, 7 ) - SUM*T7
          C( J, 8 ) = C( J, 8 ) - SUM*T8
          C( J, 9 ) = C( J, 9 ) - SUM*T9
          C( J, 10 ) = C( J, 10 ) - SUM*T10
400      CONTINUE
          GO TO 410
      END IF
410 CONTINUE
      RETURN
*
*      End of DLARFX
*
      END

```

— LAPACK dlarfx —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlarfx (side m n v tau c ldc work)
    (declare (type (double-float) tau)
              (type (simple-array double-float (*)) work c v)
              (type fixnum ldc n m)
              (type character side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (v double-float v-%data% v-%offset%))

```



```

(c double-float c-%data% c-%offset%)
(work double-float work-%data% work-%offset%))
(prog ((sum 0.0) (t1 0.0) (t10 0.0) (t2 0.0) (t3 0.0) (t4 0.0) (t5 0.0)
      (t6 0.0) (t7 0.0) (t8 0.0) (t9 0.0) (v1 0.0) (v10 0.0) (v2 0.0)
      (v3 0.0) (v4 0.0) (v5 0.0) (v6 0.0) (v7 0.0) (v8 0.0) (v9 0.0)
      (j 0))
(declare (type (double-float) sum t1 t10 t2 t3 t4 t5 t6 t7 t8 t9 v1 v10
              v2 v3 v4 v5 v6 v7 v8 v9)
          (type fixnum j))
(if (= tau zero) (go end_label))
(cond
 ((char-equal side #\L)
  (tagbody
   (f2cl-lib:computed-goto
    (label10 label130 label150 label170 label190 label110 label1130
     label150 label170 label190)
    m)
   (dgemv "Transpose" m n one c ldc v 1 zero work 1)
   (dger m n (- tau) v 1 work 1 c ldc)
   (go end_label)

label10
   (setf t1
        (+ one
          (* (- tau)
             (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
             (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 (> j n) nil)
   (tagbody
    (setf (f2cl-lib:fref c-%data%
                        (1 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* t1
            (f2cl-lib:fref c-%data%
                          (1 j)
                          ((1 ldc) (1 *))
                          c-%offset%))))
    (go end_label)

label130
   (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
   (setf t1 (* tau v1))
   (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
   (setf t2 (* tau v2))
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                 (> j n) nil)
   (tagbody
    (setf sum
          (+
            (* v1

```

```

(f2cl-lib:fref c-%data%
  (1 j)
  ((1 ldc) (1 *))
  c-%offset%))
(* v2
  (f2cl-lib:fref c-%data%
    (2 j)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (1 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (1 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (2 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (2 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t2))))
(go end_label)

label50
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (1 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (2 j)

```

```

((1 ldc) (1 *))
c-%offset%))
(* v3
  (f2cl-lib:fref c-%data%
    (3 j)
    ((1 ldc) (1 *))
    c-%offset%))))
(setf (f2cl-lib:fref c-%data%
  (1 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (1 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (2 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (2 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (3 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (3 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t3))))
(go end_label)

label70
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody

```

```

(setf sum
  (+
    (* v1
      (f2cl-lib:fref c-%data%
                     (1 j)
                     ((1 ldc) (1 *)))
      c-%offset%))
    (* v2
      (f2cl-lib:fref c-%data%
                     (2 j)
                     ((1 ldc) (1 *)))
      c-%offset%))
    (* v3
      (f2cl-lib:fref c-%data%
                     (3 j)
                     ((1 ldc) (1 *)))
      c-%offset%))
    (* v4
      (f2cl-lib:fref c-%data%
                     (4 j)
                     ((1 ldc) (1 *)))
      c-%offset%))))
(setf (f2cl-lib:fref c-%data%
                    (1 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
(-
  (f2cl-lib:fref c-%data%
                 (1 j)
                 ((1 ldc) (1 *)))
  c-%offset%)
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                    (2 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
(-
  (f2cl-lib:fref c-%data%
                 (2 j)
                 ((1 ldc) (1 *)))
  c-%offset%)
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                    (3 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
(-
  (f2cl-lib:fref c-%data%
                 (3 j)
                 ((1 ldc) (1 *)))
  c-%offset%)
  (* sum t3)))

```

```

                                c-%offset%)
      (* sum t3)))
    (setf (f2cl-lib:fref c-%data%
                        (4 j)
                        ((1 ldc) (1 *)))
          c-%offset%)
    (-
      (f2cl-lib:fref c-%data%
                    (4 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
    (* sum t4))))))
  (go end_label)

label90
  (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
  (setf t1 (* tau v1))
  (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
  (setf t2 (* tau v2))
  (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
  (setf t3 (* tau v3))
  (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
  (setf t4 (* tau v4))
  (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
  (setf t5 (* tau v5))
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (setf sum
      (+
        (* v1
          (f2cl-lib:fref c-%data%
                        (1 j)
                        ((1 ldc) (1 *)))
          c-%offset%))
        (* v2
          (f2cl-lib:fref c-%data%
                        (2 j)
                        ((1 ldc) (1 *)))
          c-%offset%))
        (* v3
          (f2cl-lib:fref c-%data%
                        (3 j)
                        ((1 ldc) (1 *)))
          c-%offset%))
        (* v4
          (f2cl-lib:fref c-%data%
                        (4 j)
                        ((1 ldc) (1 *)))
          c-%offset%))
        (* v5

```

```

(f2cl-lib:fref c-%data%
  (5 j)
  ((1 ldc) (1 *))
  c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (1 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (1 j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (2 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (2 j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (3 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (3 j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (4 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (4 j)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t4)))
(setf (f2cl-lib:fref c-%data%
  (5 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%

```

```

                                (5 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
                                (* sum t5))))
    (go end_label)
label110
    (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
    (setf t1 (* tau v1))
    (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
    (setf t2 (* tau v2))
    (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
    (setf t3 (* tau v3))
    (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
    (setf t4 (* tau v4))
    (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
    (setf t5 (* tau v5))
    (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
    (setf t6 (* tau v6))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (setf sum
        (+
          (* v1
            (f2cl-lib:fref c-%data%
                          (1 j)
                          ((1 ldc) (1 *))
                          c-%offset%))
          (* v2
            (f2cl-lib:fref c-%data%
                          (2 j)
                          ((1 ldc) (1 *))
                          c-%offset%))
          (* v3
            (f2cl-lib:fref c-%data%
                          (3 j)
                          ((1 ldc) (1 *))
                          c-%offset%))
          (* v4
            (f2cl-lib:fref c-%data%
                          (4 j)
                          ((1 ldc) (1 *))
                          c-%offset%))
          (* v5
            (f2cl-lib:fref c-%data%
                          (5 j)
                          ((1 ldc) (1 *))
                          c-%offset%))
          (* v6
            (f2cl-lib:fref c-%data%

```

```

(6 j)
((1 ldc) (1 *))
c-%offset%)))
(setf (f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t1)))
(setf (f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t2)))
(setf (f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(5 j)

```



```

                                ((1 ldc) (1 *))
                                c-%offset%)

        (* sum t5)))
(setf (f2cl-lib:fref c-%data%
                    (6 j)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                        (6 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (* sum t6))))))
label130 (go end_label)

(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j n) nil)

(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
                        (1 j)
                        ((1 ldc) (1 *))
                        c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
                        (2 j)
                        ((1 ldc) (1 *))
                        c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
                        (3 j)
                        ((1 ldc) (1 *))
                        c-%offset%))
      (* v4
```

```

(f2cl-lib:fref c-%data%
  (4 j)
  ((1 ldc) (1 *))
  c-%offset%))
(* v5
  (f2cl-lib:fref c-%data%
    (5 j)
    ((1 ldc) (1 *))
    c-%offset%))
(* v6
  (f2cl-lib:fref c-%data%
    (6 j)
    ((1 ldc) (1 *))
    c-%offset%))
(* v7
  (f2cl-lib:fref c-%data%
    (7 j)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (1 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (1 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (2 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (2 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (3 j)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (3 j)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t3)))
(setf (f2cl-lib:fref c-%data%

```

```

(4 j)
((1 ldc) (1 *))
c-%offset%)

(-
(f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%)

(-
(f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t5)))
(setf (f2cl-lib:fref c-%data%
(6 j)
((1 ldc) (1 *))
c-%offset%)

(-
(f2cl-lib:fref c-%data%
(6 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t6)))
(setf (f2cl-lib:fref c-%data%
(7 j)
((1 ldc) (1 *))
c-%offset%)

(-
(f2cl-lib:fref c-%data%
(7 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t7))))))
(go end_label)

label150
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))

```

```

(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (1 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (2 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (3 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (4 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%
          (5 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v6
        (f2cl-lib:fref c-%data%
          (6 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v7
        (f2cl-lib:fref c-%data%
          (7 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v8
        (f2cl-lib:fref c-%data%
          (8 j)

```

```

((1 ldc) (1 *))
c-%offset%)))
(setf (f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t1)))
(setf (f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t2)))
(setf (f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))

```

```

                                c-%offset%)
      (* sum t5)))
    (setf (f2cl-lib:fref c-%data%
                        (6 j)
                        ((1 ldc) (1 *)))
          c-%offset%)
    (-
      (f2cl-lib:fref c-%data%
                    (6 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
    (* sum t6)))
  (setf (f2cl-lib:fref c-%data%
                    (7 j)
                    ((1 ldc) (1 *)))
        c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
                  (7 j)
                  ((1 ldc) (1 *)))
    c-%offset%)
  (* sum t7)))
  (setf (f2cl-lib:fref c-%data%
                    (8 j)
                    ((1 ldc) (1 *)))
        c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
                  (8 j)
                  ((1 ldc) (1 *)))
    c-%offset%)
  (* sum t8))))
  (go end_label)
label170
  (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
  (setf t1 (* tau v1))
  (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
  (setf t2 (* tau v2))
  (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
  (setf t3 (* tau v3))
  (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
  (setf t4 (* tau v4))
  (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
  (setf t5 (* tau v5))
  (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
  (setf t6 (* tau v6))
  (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
  (setf t7 (* tau v7))
  (setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
  (setf t8 (* tau v8))

```

```

(setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
(setf t9 (* tau v9))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (1 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (2 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (3 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (4 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%
          (5 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v6
        (f2cl-lib:fref c-%data%
          (6 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v7
        (f2cl-lib:fref c-%data%
          (7 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v8
        (f2cl-lib:fref c-%data%
          (8 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v9
        (f2cl-lib:fref c-%data%
          (9 j)

```

```

((1 ldc) (1 *))
c-%offset%)))
(setf (f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(1 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t1)))
(setf (f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(2 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t2)))
(setf (f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(3 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%))
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%))
(-
(f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))

```



```

                                c-%offset%)
      (* sum t5)))
    (setf (f2cl-lib:fref c-%data%
                        (6 j)
                        ((1 ldc) (1 *)))
          c-%offset%)
    (-
      (f2cl-lib:fref c-%data%
                    (6 j)
                    ((1 ldc) (1 *)))
      c-%offset%)
    (* sum t6)))
  (setf (f2cl-lib:fref c-%data%
                      (7 j)
                      ((1 ldc) (1 *)))
        c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
                  (7 j)
                  ((1 ldc) (1 *)))
    c-%offset%)
  (* sum t7)))
  (setf (f2cl-lib:fref c-%data%
                      (8 j)
                      ((1 ldc) (1 *)))
        c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
                  (8 j)
                  ((1 ldc) (1 *)))
    c-%offset%)
  (* sum t8)))
  (setf (f2cl-lib:fref c-%data%
                      (9 j)
                      ((1 ldc) (1 *)))
        c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
                  (9 j)
                  ((1 ldc) (1 *)))
    c-%offset%)
  (* sum t9))))))
  (go end_label)

label190
  (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
  (setf t1 (* tau v1))
  (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
  (setf t2 (* tau v2))
  (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
  (setf t3 (* tau v3))

```

```

(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
(setf t9 (* tau v9))
(setf v10 (f2cl-lib:fref v-%data% (10) ((1 *)) v-%offset%))
(setf t10 (* tau v10))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (1 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (2 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (3 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (4 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%
          (5 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v6
        (f2cl-lib:fref c-%data%
          (6 j)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v7

```

```

        (f2cl-lib:fref c-%data%
                     (7 j)
                     ((1 ldc) (1 *))
                     c-%offset%))
(* v8
  (f2cl-lib:fref c-%data%
               (8 j)
               ((1 ldc) (1 *))
               c-%offset%))
(* v9
  (f2cl-lib:fref c-%data%
               (9 j)
               ((1 ldc) (1 *))
               c-%offset%))
(* v10
  (f2cl-lib:fref c-%data%
               (10 j)
               ((1 ldc) (1 *))
               c-%offset%))))
(setf (f2cl-lib:fref c-%data%
                  (1 j)
                  ((1 ldc) (1 *))
                  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
                (1 j)
                ((1 ldc) (1 *))
                c-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                  (2 j)
                  ((1 ldc) (1 *))
                  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
                (2 j)
                ((1 ldc) (1 *))
                c-%offset%))
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                  (3 j)
                  ((1 ldc) (1 *))
                  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
                (3 j)
                ((1 ldc) (1 *))
                c-%offset%))
  (* sum t3)))
(setf (f2cl-lib:fref c-%data%

```

```

(4 j)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(4 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(5 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t5)))
(setf (f2cl-lib:fref c-%data%
(6 j)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(6 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t6)))
(setf (f2cl-lib:fref c-%data%
(7 j)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(7 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t7)))
(setf (f2cl-lib:fref c-%data%
(8 j)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(8 j)
((1 ldc) (1 *))
c-%offset%)
(* sum t8)))
(setf (f2cl-lib:fref c-%data%

```

```

                                (9 j)
                                ((1 ldc) (1 *))
                                c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                      (9 j)
                      ((1 ldc) (1 *))
                      c-%offset%)
        (* sum t9)))
      (setf (f2cl-lib:fref c-%data%
                        (10 j)
                        ((1 ldc) (1 *))
                        c-%offset%)
            (-
              (f2cl-lib:fref c-%data%
                            (10 j)
                            ((1 ldc) (1 *))
                            c-%offset%)
              (* sum t10))))))
      (go end_label)))
(t
 (tagbody
  (f2cl-lib:computed-goto
   (label210 label230 label250 label270 label290 label310 label330
    label350 label370 label390)
   n)
  (dgemv "No transpose" m n one c ldc v 1 zero work 1)
  (dger m n (- tau) work 1 v 1 c ldc)
  (go end_label)
label210
  (setf t1
    (+ one
      (* (- tau)
        (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
        (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))))))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j m) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data%
                          (j 1)
                          ((1 ldc) (1 *))
                          c-%offset%)
            (* t1
              (f2cl-lib:fref c-%data%
                            (j 1)
                            ((1 ldc) (1 *))
                            c-%offset%))))))
      (go end_label)
label230
  (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))

```

```

(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
      (j 1)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%)
        (* sum t1)))
    (setf (f2cl-lib:fref c-%data%
      (j 2)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%)
        (* sum t2))))))
(go end_label)
label250
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)
(tagbody
  (setf sum

```

```

      (+
      (* v1
        (f2cl-lib:fref c-%data%
                       (j 1)
                       ((1 ldc) (1 *)))
        c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
                       (j 2)
                       ((1 ldc) (1 *)))
        c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
                       (j 3)
                       ((1 ldc) (1 *)))
        c-%offset%))))
      (setf (f2cl-lib:fref c-%data%
                        (j 1)
                        ((1 ldc) (1 *)))
            c-%offset%)
      (-
      (f2cl-lib:fref c-%data%
                    (j 1)
                    ((1 ldc) (1 *)))
      c-%offset%)
      (* sum t1)))
      (setf (f2cl-lib:fref c-%data%
                        (j 2)
                        ((1 ldc) (1 *)))
            c-%offset%)
      (-
      (f2cl-lib:fref c-%data%
                    (j 2)
                    ((1 ldc) (1 *)))
      c-%offset%)
      (* sum t2)))
      (setf (f2cl-lib:fref c-%data%
                        (j 3)
                        ((1 ldc) (1 *)))
            c-%offset%)
      (-
      (f2cl-lib:fref c-%data%
                    (j 3)
                    ((1 ldc) (1 *)))
      c-%offset%)
      (* sum t3))))))
      (go end_label)
label270
      (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
      (setf t1 (* tau v1))

```

```

(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (j 3)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (j 4)
          ((1 ldc) (1 *))
          c-%offset%))))
    (setf (f2cl-lib:fref c-%data%
      (j 1)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%)
        (* sum t1)))
    (setf (f2cl-lib:fref c-%data%
      (j 2)
      ((1 ldc) (1 *))
      c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%)

```



```

        (* sum t2)))
      (setf (f2cl-lib:fref c-%data%
                          (j 3)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 3)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t3)))
      (setf (f2cl-lib:fref c-%data%
                          (j 4)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 4)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t4))))))
      (go end_label)
label290
      (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
      (setf t1 (* tau v1))
      (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
      (setf t2 (* tau v2))
      (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
      (setf t3 (* tau v3))
      (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
      (setf t4 (* tau v4))
      (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
      (setf t5 (* tau v5))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j m) nil)
      (tagbody
        (setf sum
          (+
            (* v1
              (f2cl-lib:fref c-%data%
                            (j 1)
                            ((1 ldc) (1 *))
                            c-%offset%))
            (* v2
              (f2cl-lib:fref c-%data%
                            (j 2)
                            ((1 ldc) (1 *))
                            c-%offset%))
            (* v3
              (f2cl-lib:fref c-%data%
                            (j 3)
                            ((1 ldc) (1 *))
                            c-%offset%))
            (* v4
              (f2cl-lib:fref c-%data%
                            (j 4)
                            ((1 ldc) (1 *))
                            c-%offset%))
            (* v5
              (f2cl-lib:fref c-%data%
                            (j 5)
                            ((1 ldc) (1 *))
                            c-%offset%))
            t1
            t2
            t3
            t4
            t5
            sum)))
        (incf j)
        (if (= j m)
            (return sum)
            (tagbody)))

```

```

                                (j 3)
                                ((1 ldc) (1 *))
                                c-%offset%)
(* v4
  (f2cl-lib:fref c-%data%
    (j 4)
    ((1 ldc) (1 *))
    c-%offset%))
(* v5
  (f2cl-lib:fref c-%data%
    (j 5)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (j 1)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 1)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (j 2)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 2)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (j 3)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 3)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (j 4)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 4)

```

```

                                ((1 ldc) (1 *))
                                c-%offset%)
                                (* sum t4)))
(setf (f2cl-lib:fref c-%data%
                    (j 5)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                      (j 5)
                      ((1 ldc) (1 *))
                      c-%offset%)
        (* sum t5))))))
(go end_label)

label310
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j m) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
                      (j 1)
                      ((1 ldc) (1 *))
                      c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
                      (j 2)
                      ((1 ldc) (1 *))
                      c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
                      (j 3)
                      ((1 ldc) (1 *))
                      c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
                      (j 4)

```

```

((1 ldc) (1 *))
c-%offset%))
(* v5
  (f2cl-lib:fref c-%data%
    (j 5)
    ((1 ldc) (1 *))
    c-%offset%))
(* v6
  (f2cl-lib:fref c-%data%
    (j 6)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (j 1)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 1)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (j 2)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 2)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (j 3)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 3)
    ((1 ldc) (1 *))
    c-%offset%))
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (j 4)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 4)
    ((1 ldc) (1 *))

```

```

                                c-%offset%)
      (* sum t4)))
    (setf (f2cl-lib:fref c-%data%
                        (j 5)
                        ((1 ldc) (1 *))
                        c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                      (j 5)
                      ((1 ldc) (1 *))
                      c-%offset%)
        (* sum t5)))
    (setf (f2cl-lib:fref c-%data%
                        (j 6)
                        ((1 ldc) (1 *))
                        c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                      (j 6)
                      ((1 ldc) (1 *))
                      c-%offset%)
        (* sum t6))))))
  (go end_label)

label330
  (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
  (setf t1 (* tau v1))
  (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
  (setf t2 (* tau v2))
  (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
  (setf t3 (* tau v3))
  (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
  (setf t4 (* tau v4))
  (setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
  (setf t5 (* tau v5))
  (setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
  (setf t6 (* tau v6))
  (setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
  (setf t7 (* tau v7))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j m) nil)
  (tagbody
    (setf sum
      (+
        (* v1
          (f2cl-lib:fref c-%data%
                        (j 1)
                        ((1 ldc) (1 *))
                        c-%offset%))
        (* v2
          (f2cl-lib:fref c-%data%

```

```

                                (j 2)
                                ((1 ldc) (1 *))
                                c-%offset%)
(* v3
  (f2cl-lib:fref c-%data%
                (j 3)
                ((1 ldc) (1 *))
                c-%offset%))
(* v4
  (f2cl-lib:fref c-%data%
                (j 4)
                ((1 ldc) (1 *))
                c-%offset%))
(* v5
  (f2cl-lib:fref c-%data%
                (j 5)
                ((1 ldc) (1 *))
                c-%offset%))
(* v6
  (f2cl-lib:fref c-%data%
                (j 6)
                ((1 ldc) (1 *))
                c-%offset%))
(* v7
  (f2cl-lib:fref c-%data%
                (j 7)
                ((1 ldc) (1 *))
                c-%offset%))))
(setf (f2cl-lib:fref c-%data%
                    (j 1)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                        (j 1)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                    (j 2)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                        (j 2)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                    (j 3)

```

```

((1 ldc) (1 *))
c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 3)
    ((1 ldc) (1 *))
    c-%offset%)
    (* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (j 4)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 4)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t4)))
(setf (f2cl-lib:fref c-%data%
  (j 5)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 5)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t5)))
(setf (f2cl-lib:fref c-%data%
  (j 6)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 6)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t6)))
(setf (f2cl-lib:fref c-%data%
  (j 7)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 7)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t7))))
(go end_label)
label350

```

```

(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)
(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (j 3)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (j 4)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%
          (j 5)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v6
        (f2cl-lib:fref c-%data%
          (j 6)
          ((1 ldc) (1 *))

```



```

                                c-%offset%))
(* v7
  (f2cl-lib:fref c-%data%
    (j 7)
    ((1 ldc) (1 *))
    c-%offset%))
(* v8
  (f2cl-lib:fref c-%data%
    (j 8)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (j 1)
  ((1 ldc) (1 *))
  c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 1)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (j 2)
  ((1 ldc) (1 *))
  c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 2)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (j 3)
  ((1 ldc) (1 *))
  c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 3)
    ((1 ldc) (1 *))
    c-%offset%)
  (* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (j 4)
  ((1 ldc) (1 *))
  c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 4)
    ((1 ldc) (1 *))
    c-%offset%)

```

```

        (* sum t4)))
      (setf (f2cl-lib:fref c-%data%
                          (j 5)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 5)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t5)))
      (setf (f2cl-lib:fref c-%data%
                          (j 6)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 6)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t6)))
      (setf (f2cl-lib:fref c-%data%
                          (j 7)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 7)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t7)))
      (setf (f2cl-lib:fref c-%data%
                          (j 8)
                          ((1 ldc) (1 *))
                          c-%offset%)
        (-
          (f2cl-lib:fref c-%data%
                        (j 8)
                        ((1 ldc) (1 *))
                        c-%offset%)
          (* sum t8))))
      (go end_label)
label370
      (setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
      (setf t1 (* tau v1))
      (setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
      (setf t2 (* tau v2))
      (setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
      (setf t3 (* tau v3))
      (setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))

```

```

(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
(setf t9 (* tau v9))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)

(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (j 3)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (j 4)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%
          (j 5)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v6
        (f2cl-lib:fref c-%data%
          (j 6)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v7
        (f2cl-lib:fref c-%data%
          (j 7)
          ((1 ldc) (1 *))

```

```

                                c-%offset%))
(* v8
  (f2cl-lib:fref c-%data%
    (j 8)
    ((1 ldc) (1 *))
    c-%offset%))
(* v9
  (f2cl-lib:fref c-%data%
    (j 9)
    ((1 ldc) (1 *))
    c-%offset%)))
(setf (f2cl-lib:fref c-%data%
  (j 1)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 1)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t1)))
(setf (f2cl-lib:fref c-%data%
  (j 2)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 2)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t2)))
(setf (f2cl-lib:fref c-%data%
  (j 3)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 3)
    ((1 ldc) (1 *))
    c-%offset%))
  (* sum t3)))
(setf (f2cl-lib:fref c-%data%
  (j 4)
  ((1 ldc) (1 *))
  c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 4)
    ((1 ldc) (1 *))
    c-%offset%))

```

```

      (* sum t4)))
(setf (f2cl-lib:fref c-%data%
      (j 5)
      ((1 ldc) (1 *))
      c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 5)
    ((1 ldc) (1 *))
    c-%offset%))
      (* sum t5)))
(setf (f2cl-lib:fref c-%data%
      (j 6)
      ((1 ldc) (1 *))
      c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 6)
    ((1 ldc) (1 *))
    c-%offset%))
      (* sum t6)))
(setf (f2cl-lib:fref c-%data%
      (j 7)
      ((1 ldc) (1 *))
      c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 7)
    ((1 ldc) (1 *))
    c-%offset%))
      (* sum t7)))
(setf (f2cl-lib:fref c-%data%
      (j 8)
      ((1 ldc) (1 *))
      c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 8)
    ((1 ldc) (1 *))
    c-%offset%))
      (* sum t8)))
(setf (f2cl-lib:fref c-%data%
      (j 9)
      ((1 ldc) (1 *))
      c-%offset%))
(-
  (f2cl-lib:fref c-%data%
    (j 9)
    ((1 ldc) (1 *))
    c-%offset%))

```

```

(* sum t9))))))
(go end_label)

label390
(setf v1 (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%))
(setf t1 (* tau v1))
(setf v2 (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%))
(setf t2 (* tau v2))
(setf v3 (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%))
(setf t3 (* tau v3))
(setf v4 (f2cl-lib:fref v-%data% (4) ((1 *)) v-%offset%))
(setf t4 (* tau v4))
(setf v5 (f2cl-lib:fref v-%data% (5) ((1 *)) v-%offset%))
(setf t5 (* tau v5))
(setf v6 (f2cl-lib:fref v-%data% (6) ((1 *)) v-%offset%))
(setf t6 (* tau v6))
(setf v7 (f2cl-lib:fref v-%data% (7) ((1 *)) v-%offset%))
(setf t7 (* tau v7))
(setf v8 (f2cl-lib:fref v-%data% (8) ((1 *)) v-%offset%))
(setf t8 (* tau v8))
(setf v9 (f2cl-lib:fref v-%data% (9) ((1 *)) v-%offset%))
(setf t9 (* tau v9))
(setf v10 (f2cl-lib:fref v-%data% (10) ((1 *)) v-%offset%))
(setf t10 (* tau v10))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j m) nil)

(tagbody
  (setf sum
    (+
      (* v1
        (f2cl-lib:fref c-%data%
          (j 1)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v2
        (f2cl-lib:fref c-%data%
          (j 2)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v3
        (f2cl-lib:fref c-%data%
          (j 3)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v4
        (f2cl-lib:fref c-%data%
          (j 4)
          ((1 ldc) (1 *))
          c-%offset%))
      (* v5
        (f2cl-lib:fref c-%data%

```

```

                                (j 5)
                                ((1 ldc) (1 *))
                                c-%offset%)
(* v6
  (f2cl-lib:fref c-%data%
                (j 6)
                ((1 ldc) (1 *))
                c-%offset%))
(* v7
  (f2cl-lib:fref c-%data%
                (j 7)
                ((1 ldc) (1 *))
                c-%offset%))
(* v8
  (f2cl-lib:fref c-%data%
                (j 8)
                ((1 ldc) (1 *))
                c-%offset%))
(* v9
  (f2cl-lib:fref c-%data%
                (j 9)
                ((1 ldc) (1 *))
                c-%offset%))
(* v10
  (f2cl-lib:fref c-%data%
                (j 10)
                ((1 ldc) (1 *))
                c-%offset%)))
(setf (f2cl-lib:fref c-%data%
                    (j 1)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                        (j 1)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (* sum t1)))
(setf (f2cl-lib:fref c-%data%
                    (j 2)
                    ((1 ldc) (1 *))
                    c-%offset%)
      (-
        (f2cl-lib:fref c-%data%
                        (j 2)
                        ((1 ldc) (1 *))
                        c-%offset%)
        (* sum t2)))
(setf (f2cl-lib:fref c-%data%
                    (j 3)

```

```

((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(j 3)
((1 ldc) (1 *))
c-%offset%)
(* sum t3)))
(setf (f2cl-lib:fref c-%data%
(j 4)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(j 4)
((1 ldc) (1 *))
c-%offset%)
(* sum t4)))
(setf (f2cl-lib:fref c-%data%
(j 5)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(j 5)
((1 ldc) (1 *))
c-%offset%)
(* sum t5)))
(setf (f2cl-lib:fref c-%data%
(j 6)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(j 6)
((1 ldc) (1 *))
c-%offset%)
(* sum t6)))
(setf (f2cl-lib:fref c-%data%
(j 7)
((1 ldc) (1 *))
c-%offset%)
(-
(f2cl-lib:fref c-%data%
(j 7)
((1 ldc) (1 *))
c-%offset%)
(* sum t7)))
(setf (f2cl-lib:fref c-%data%
(j 8)

```



```

((1 ldc) (1 *))
c-%offset%)
(-
  (f2cl-lib:fref c-%data%
    (j 8)
    ((1 ldc) (1 *))
    c-%offset%)
    (* sum t8)))
(setf (f2cl-lib:fref c-%data%
  (j 9)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 9)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t9)))
(setf (f2cl-lib:fref c-%data%
  (j 10)
  ((1 ldc) (1 *))
  c-%offset%)
  (-
    (f2cl-lib:fref c-%data%
      (j 10)
      ((1 ldc) (1 *))
      c-%offset%)
      (* sum t10))))))
(go end_label))))
end_label
(return (values nil nil nil nil nil nil nil nil))))))

```

dlartg LAPACK

— dlartg.input —

```

)set break resume
)sys rm -f dlartg.output
)spool dlartg.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— dlartg.help —

=====

dlartg examples

=====

=====

Man Page Details

=====

NAME

DLARTG - a plane rotation so that [CS SN]

SYNOPSIS

SUBROUTINE DLARTG(F, G, CS, SN, R)

DOUBLE PRECISION CS, F, G, R, SN

Purpose

=====

DLARTG generate a plane rotation so that

$$\begin{bmatrix} CS & SN \\ -SN & CS \end{bmatrix} \cdot \begin{bmatrix} F \\ G \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{where } CS^2 + SN^2 = 1.$$

This is a slower, more accurate version of the BLAS1 routine DROTG, with the following other differences:

F and G are unchanged on return.

If G=0, then CS=1 and SN=0.

If F=0 and (G .ne. 0), then CS=0 and SN=1 without doing any floating point operations (saves work in DBDSQR when there are zeros on the diagonal).

If F exceeds G in magnitude, CS will be positive.

Arguments

=====

F (input) DOUBLE PRECISION
The first component of vector to be rotated.

G (input) DOUBLE PRECISION
The second component of vector to be rotated.

CS (output) DOUBLE PRECISION
 The cosine of the rotation.

SN (output) DOUBLE PRECISION
 The sine of the rotation.

R (output) DOUBLE PRECISION
 The nonzero component of the rotated vector.

— dlartg.f —

```

SUBROUTINE DLARTG( F, G, CS, SN, R )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    September 30, 1994
*
*    .. Scalar Arguments ..
*    DOUBLE PRECISION   CS, F, G, R, SN
*    ..
*
*    =====
*
*    .. Parameters ..
*    DOUBLE PRECISION   ZERO
*    PARAMETER          ( ZERO = 0.0D0 )
*    DOUBLE PRECISION   ONE
*    PARAMETER          ( ONE = 1.0D0 )
*    DOUBLE PRECISION   TWO
*    PARAMETER          ( TWO = 2.0D0 )
*    ..
*    .. Local Scalars ..
*    LOGICAL            FIRST
*    INTEGER            COUNT, I
*    DOUBLE PRECISION   EPS, F1, G1, SAFMIN, SAFMN2, SAFMX2, SCALE
*    ..
*    .. External Functions ..
*    DOUBLE PRECISION   DLAMCH
*    EXTERNAL           DLAMCH
*    ..
*    .. Intrinsic Functions ..
*    INTRINSIC          ABS, INT, LOG, MAX, SQRT
*    ..
*    .. Save statement ..
*    SAVE               FIRST, SAFMX2, SAFMIN, SAFMN2

```

```

*      ..
*      .. Data statements ..
DATA          FIRST / .TRUE. /
*      ..
*      .. Executable Statements ..
*
      IF( FIRST ) THEN
        FIRST = .FALSE.
        SAFMIN = DLAMCH( 'S' )
        EPS = DLAMCH( 'E' )
        SAFMN2 = DLAMCH( 'B' )**INT( LOG( SAFMIN / EPS ) /
$      LOG( DLAMCH( 'B' ) ) / TWO )
        SAFMX2 = ONE / SAFMN2
      END IF
      IF( G.EQ.ZERO ) THEN
        CS = ONE
        SN = ZERO
        R = F
      ELSE IF( F.EQ.ZERO ) THEN
        CS = ZERO
        SN = ONE
        R = G
      ELSE
        F1 = F
        G1 = G
        SCALE = MAX( ABS( F1 ), ABS( G1 ) )
        IF( SCALE.GE.SAFMX2 ) THEN
          COUNT = 0
10      CONTINUE
          COUNT = COUNT + 1
          F1 = F1*SAFMN2
          G1 = G1*SAFMN2
          SCALE = MAX( ABS( F1 ), ABS( G1 ) )
          IF( SCALE.GE.SAFMX2 )
$            GO TO 10
          R = SQRT( F1**2+G1**2 )
          CS = F1 / R
          SN = G1 / R
          DO 20 I = 1, COUNT
            R = R*SAFMX2
20      CONTINUE
          ELSE IF( SCALE.LE.SAFMN2 ) THEN
            COUNT = 0
30      CONTINUE
            COUNT = COUNT + 1
            F1 = F1*SAFMX2
            G1 = G1*SAFMX2
            SCALE = MAX( ABS( F1 ), ABS( G1 ) )
            IF( SCALE.LE.SAFMN2 )
$              GO TO 30

```

```

        R = SQRT( F1**2+G1**2 )
        CS = F1 / R
        SN = G1 / R
        DO 40 I = 1, COUNT
            R = R*SAFMN2
40      CONTINUE
    ELSE
        R = SQRT( F1**2+G1**2 )
        CS = F1 / R
        SN = G1 / R
    END IF
    IF( ABS( F ).GT.ABS( G ) .AND. CS.LT.ZERO ) THEN
        CS = -CS
        SN = -SN
        R = -R
    END IF
    END IF
    RETURN
*
*   End of DLARTG
*
    END

```

— LAPACK dlartg —

```

(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two))
  (let ((safmx2 0.0) (safmin 0.0) (safmn2 0.0) (first$ nil))
    (declare (type (member t nil) first$)
              (type (double-float) safmn2 safmin safmx2))
    (setq first$ t)
    (defun dlartg (f g cs sn r)
      (declare (type (double-float) r sn cs g f))
      (prog ((eps 0.0) (f1 0.0) (g1 0.0) (scale 0.0) (i 0) (count$ 0))
        (declare (type (double-float) eps f1 g1 scale)
                  (type fixnum count$ i))
        (cond
         (first$
          (setf first$ nil)
          (setf safmin (dlamch "S"))
          (setf eps (dlamch "E"))
          (setf safmn2
                 (expt (dlamch "B")
                        (f2c1-lib:int

```

```

                                (/
                                (/ (f2cl-lib:flog (/ safmin eps))
                                (f2cl-lib:flog (dlamch "B")))
                                two))))
    (setf safmx2 (/ one safmn2))))
(cond
  ((= g zero)
   (setf cs one)
   (setf sn zero)
   (setf r f))
  ((= f zero)
   (setf cs zero)
   (setf sn one)
   (setf r g))
  (t
   (setf f1 f)
   (setf g1 g)
   (setf scale (max (abs f1) (abs g1)))
   (cond
    ((>= scale safmx2)
     (tagbody
      (setf count$ 0)
label110
      (setf count$ (f2cl-lib:int-add count$ 1))
      (setf f1 (* f1 safmn2))
      (setf g1 (* g1 safmn2))
      (setf scale (max (abs f1) (abs g1)))
      (if (>= scale safmx2) (go label110))
      (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
      (setf cs (/ f1 r))
      (setf sn (/ g1 r))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i count$) nil)
        (tagbody (setf r (* r safmx2)) label120))))
    ((<= scale safmn2)
     (tagbody
      (setf count$ 0)
label130
      (setf count$ (f2cl-lib:int-add count$ 1))
      (setf f1 (* f1 safmx2))
      (setf g1 (* g1 safmx2))
      (setf scale (max (abs f1) (abs g1)))
      (if (<= scale safmn2) (go label130))
      (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
      (setf cs (/ f1 r))
      (setf sn (/ g1 r))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i count$) nil)
        (tagbody (setf r (* r safmn2)) label140))))
   (t

```

```

      (setf r (f2cl-lib:fsqrt (+ (expt f1 2) (expt g1 2))))
      (setf cs (/ f1 r))
      (setf sn (/ g1 r)))
    (cond
      ((and (> (abs f) (abs g)) (< cs zero))
       (setf cs (- cs))
       (setf sn (- sn))
       (setf r (- r))))))
  end_label
  (return (values nil nil cs sn r))))))

```

dlas2 LAPACK

— dlas2.input —

```

)set break resume
)sys rm -f dlas2.output
)spool dlas2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlas2.help —

```

=====
dlas2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLAS2 - the singular values of the 2-by-2 matrix $\begin{bmatrix} F & G \\ 0 & H \end{bmatrix}$

SYNOPSIS

SUBROUTINE DLAS2(F, G, H, SSMIN, SSMAX)

DOUBLE PRECISION F, G, H, SSMAX, SSMIN

PURPOSE

DLAS2 computes the singular values of the 2-by-2 matrix

$$\begin{bmatrix} F & G \\ 0 & H \end{bmatrix}.$$
 On return, SSMIN is the smaller singular value and SSMAX is the larger singular value.

ARGUMENTS

F (input) DOUBLE PRECISION
 The (1,1) element of the 2-by-2 matrix.

G (input) DOUBLE PRECISION
 The (1,2) element of the 2-by-2 matrix.

H (input) DOUBLE PRECISION
 The (2,2) element of the 2-by-2 matrix.

SSMIN (output) DOUBLE PRECISION
 The smaller singular value.

SSMAX (output) DOUBLE PRECISION
 The larger singular value.

FURTHER DETAILS

Barring over/underflow, all output quantities are correct to within a few units in the last place (ulps), even in the absence of a guard digit in addition/subtraction.

In IEEE arithmetic, the code works correctly if one matrix element is infinite.

Overflow will not occur unless the largest singular value itself overflows, or is within a few ulps of overflow. (On machines with partial overflow, like the Cray, overflow may occur if the largest singular value is within a factor of 2 of overflow.)

Underflow is harmless if underflow is gradual. Otherwise, results may correspond to a matrix modified by perturbations of size near the underflow threshold.

— dlas2.f —

SUBROUTINE DLAS2(F, G, H, SSMIN, SSMAX)

*


```

* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   September 30, 1994
*
*   .. Scalar Arguments ..
*   DOUBLE PRECISION    F, G, H, SSMAX, SSMIN
*   ..
*
*   =====
*
*   .. Parameters ..
*   DOUBLE PRECISION    ZERO
*   PARAMETER            ( ZERO = 0.0D0 )
*   DOUBLE PRECISION    ONE
*   PARAMETER            ( ONE = 1.0D0 )
*   DOUBLE PRECISION    TWO
*   PARAMETER            ( TWO = 2.0D0 )
*   ..
*   .. Local Scalars ..
*   DOUBLE PRECISION    AS, AT, AU, C, FA, FHMN, FHM, GA, HA
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC            ABS, MAX, MIN, SQRT
*   ..
*   .. Executable Statements ..
*
*   FA = ABS( F )
*   GA = ABS( G )
*   HA = ABS( H )
*   FHMN = MIN( FA, HA )
*   FHM = MAX( FA, HA )
*   IF( FHMN.EQ.ZERO ) THEN
*       SSMIN = ZERO
*       IF( FHM.EQ.ZERO ) THEN
*           SSMAX = GA
*       ELSE
*           SSMAX = MAX( FHM, GA )*SQRT( ONE+
$           ( MIN( FHM, GA ) / MAX( FHM, GA ) )**2 )
*       END IF
*   ELSE
*       IF( GA.LT.FHM ) THEN
*           AS = ONE + FHMN / FHM
*           AT = ( FHM-FHMN ) / FHM
*           AU = ( GA / FHM )**2
*           C = TWO / ( SQRT( AS*AS+AU )+SQRT( AT*AT+AU ) )
*           SSMIN = FHMN*C
*           SSMAX = FHM / C
*       ELSE
*           AU = FHM / GA

```

```

      IF( AU.EQ.ZERO ) THEN
*
*          Avoid possible harmful underflow if exponent range
*          asymmetric (true SSMIN may not underflow even if
*          AU underflows)
*
          SSMIN = ( FHMN*FHM ) / GA
          SSMAX = GA
      ELSE
          AS = ONE + FHMN / FHM
          AT = ( FHM-FHMN ) / FHM
          C = ONE / ( SQRT( ONE+( AS*AU )**2 )+
$          SQRT( ONE+( AT*AU )**2 ) )
          SSMIN = ( FHMN*C )*AU
          SSMIN = SSMIN + SSMIN
          SSMAX = GA / ( C+C )
      END IF
    END IF
  END IF
  RETURN
*
*      End of DLAS2
*
  END

```

— LAPACK dlas2 —

```

(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
    (type (double-float 1.0 1.0) one)
    (type (double-float 2.0 2.0) two))
  (defun dlas2 (f g h ssmin ssmax)
    (declare (type (double-float) ssmax ssmin h g f))
    (prog ((as 0.0) (at 0.0) (au 0.0) (c 0.0) (fa 0.0) (fhmn 0.0) (fhmx 0.0)
      (ga 0.0) (ha 0.0))
      (declare (type (double-float) as at au c fa fhmn fhmx ga ha))
      (setf fa (abs f))
      (setf ga (abs g))
      (setf ha (abs h))
      (setf fhmn (min fa ha))
      (setf fhmx (max fa ha))
      (cond
        ((= fhmn zero)
          (setf ssmin zero)
          (cond
            ((= fhmx zero)

```

```

      (setf ssmax ga))
    (t
      (setf ssmax
        (* (max fhm x ga)
          (f2cl-lib:fsqrt
            (+ one (expt (/ (min fhm x ga) (max fhm x ga) 2)))))))
    (t
      (cond
        ((< ga fhm x)
          (setf as (+ one (/ fhm n fhm x)))
          (setf at (/ (- fhm x fhm n) fhm x))
          (setf au (expt (/ ga fhm x) 2))
          (setf c
            (/ two
              (+ (f2cl-lib:fsqrt (+ (* as as) au))
                (f2cl-lib:fsqrt (+ (* at at) au)))))
          (setf ssmin (* fhm n c))
          (setf ssmax (/ fhm x c)))
        (t
          (setf au (/ fhm x ga))
          (cond
            ((= au zero)
              (setf ssmin (/ (* fhm n fhm x) ga))
              (setf ssmax ga))
            (t
              (setf as (+ one (/ fhm n fhm x)))
              (setf at (/ (- fhm x fhm n) fhm x))
              (setf c
                (/ one
                  (+ (f2cl-lib:fsqrt (+ one (expt (* as au) 2)))
                    (f2cl-lib:fsqrt (+ one (expt (* at au) 2)))))
              (setf ssmin (* fhm n c au))
              (setf ssmin (+ ssmin ssmin))
              (setf ssmax (/ ga (+ c c))))))
        (return (values nil nil nil ssmin ssmax))))

```

dlascl LAPACK

— dlascl.input —

```

)set break resume
)sys rm -f dlascl.output
)spool dlascl.output
)set message test on

```

```
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dlascl.help —

```
=====
dlascl examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASCL - the M by N real matrix A by the real scalar CTO/CFROM

SYNOPSIS

```
SUBROUTINE DLASCL( TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO )
```

| | |
|-----------|-------------------------|
| CHARACTER | TYPE |
| INTEGER | INFO, KL, KU, LDA, M, N |
| DOUBLE | PRECISION CFROM, CTO |
| DOUBLE | PRECISION A(LDA, *) |

Purpose

```
=====
```

DLASCL multiplies the M by N real matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result $CTO \cdot A(I,J)/CFROM$ does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Arguments

```
=====
```

TYPE (input) CHARACTER*1
 TYPE indices the storage type of the input matrix.
 = 'G': A is a full matrix.
 = 'L': A is a lower triangular matrix.
 = 'U': A is an upper triangular matrix.

= 'H': A is an upper Hessenberg matrix.
 = 'B': A is a symmetric band matrix with lower bandwidth KL
 and upper bandwidth KU and with the only the lower
 half stored.
 = 'Q': A is a symmetric band matrix with lower bandwidth KL
 and upper bandwidth KU and with the only the upper
 half stored.
 = 'Z': A is a band matrix with lower bandwidth KL and upper
 bandwidth KU.

KL (input) INTEGER
 The lower bandwidth of A. Referenced only if TYPE = 'B',
 'Q' or 'Z'.

KU (input) INTEGER
 The upper bandwidth of A. Referenced only if TYPE = 'B',
 'Q' or 'Z'.

CFROM (input) DOUBLE PRECISION
 CTO (input) DOUBLE PRECISION
 The matrix A is multiplied by CTO/CFROM. A(I,J) is computed
 without over/underflow if the final result CTO*A(I,J)/CFROM
 can be represented without over/underflow. CFROM must be
 nonzero.

M (input) INTEGER
 The number of rows of the matrix A. M >= 0.

N (input) INTEGER
 The number of columns of the matrix A. N >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,M)
 The matrix to be multiplied by CTO/CFROM. See TYPE for the
 storage type.

LDA (input) INTEGER
 The leading dimension of the array A. LDA >= max(1,M).

INFO (output) INTEGER
 0 - successful exit
 <0 - if INFO = -i, the i-th argument had an illegal value.

— dlascl.f —

SUBROUTINE DLASCL(TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO)

*

```

* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   February 29, 1992
*
*   .. Scalar Arguments ..
*   CHARACTER          TYPE
*   INTEGER            INFO, KL, KU, LDA, M, N
*   DOUBLE PRECISION   CFROM, CTO
*
*   ..
*   .. Array Arguments ..
*   DOUBLE PRECISION   A( LDA, * )
*   ..
*
*   =====
*
*   .. Parameters ..
*   DOUBLE PRECISION   ZERO, ONE
*   PARAMETER          ( ZERO = 0.0D0, ONE = 1.0D0 )
*
*   ..
*   .. Local Scalars ..
*   LOGICAL            DONE
*   INTEGER            I, ITYPE, J, K1, K2, K3, K4
*   DOUBLE PRECISION   BIGNUM, CFROM1, CFROMC, CTO1, CTOC, MUL, SMLNUM
*
*   ..
*   .. External Functions ..
*   LOGICAL            LSAME
*   DOUBLE PRECISION   DLAMCH
*   EXTERNAL           LSAME, DLAMCH
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC          ABS, MAX, MIN
*
*   ..
*   .. External Subroutines ..
*   EXTERNAL           XERBLA
*
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
*   INFO = 0
*
*   IF( LSAME( TYPE, 'G' ) ) THEN
*       ITYPE = 0
*   ELSE IF( LSAME( TYPE, 'L' ) ) THEN
*       ITYPE = 1
*   ELSE IF( LSAME( TYPE, 'U' ) ) THEN
*       ITYPE = 2
*   ELSE IF( LSAME( TYPE, 'H' ) ) THEN
*       ITYPE = 3

```

```

      ELSE IF( LSAME( TYPE, 'B' ) ) THEN
        ITYPE = 4
      ELSE IF( LSAME( TYPE, 'Q' ) ) THEN
        ITYPE = 5
      ELSE IF( LSAME( TYPE, 'Z' ) ) THEN
        ITYPE = 6
      ELSE
        ITYPE = -1
      END IF
*
      IF( ITYPE.EQ.-1 ) THEN
        INFO = -1
      ELSE IF( CFROM.EQ.ZERO ) THEN
        INFO = -4
      ELSE IF( M.LT.0 ) THEN
        INFO = -6
      ELSE IF( N.LT.0 .OR. ( ITYPE.EQ.4 .AND. N.NE.M ) .OR.
$      ( ITYPE.EQ.5 .AND. N.NE.M ) ) THEN
        INFO = -7
      ELSE IF( ITYPE.LE.3 .AND. LDA.LT.MAX( 1, M ) ) THEN
        INFO = -9
      ELSE IF( ITYPE.GE.4 ) THEN
        IF( KL.LT.0 .OR. KL.GT.MAX( M-1, 0 ) ) THEN
          INFO = -2
        ELSE IF( KU.LT.0 .OR. KU.GT.MAX( N-1, 0 ) .OR.
$      ( ( ITYPE.EQ.4 .OR. ITYPE.EQ.5 ) .AND. KL.NE.KU ) )
$      THEN
          INFO = -3
        ELSE IF( ( ITYPE.EQ.4 .AND. LDA.LT.KL+1 ) .OR.
$      ( ITYPE.EQ.5 .AND. LDA.LT.KU+1 ) .OR.
$      ( ITYPE.EQ.6 .AND. LDA.LT.2*KL+KU+1 ) ) THEN
          INFO = -9
        END IF
      END IF
*
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DLASCL', -INFO )
        RETURN
      END IF
*
*      Quick return if possible
*
      IF( N.EQ.0 .OR. M.EQ.0 )
$      RETURN
*
*      Get machine parameters
*
      SMLNUM = DLAMCH( 'S' )
      BIGNUM = ONE / SMLNUM
*
```

```

        CFROMC = CFROM
        CTOC = CTO
*
10  CONTINUE
    CFROM1 = CFROMC*SMLNUM
    CTO1 = CTOC / BIGNUM
    IF( ABS( CFROM1 ).GT.ABS( CTOC ) .AND. CTOC.NE.ZERO ) THEN
        MUL = SMLNUM
        DONE = .FALSE.
        CFROMC = CFROM1
    ELSE IF( ABS( CTO1 ).GT.ABS( CFROMC ) ) THEN
        MUL = BIGNUM
        DONE = .FALSE.
        CTOC = CTO1
    ELSE
        MUL = CTOC / CFROMC
        DONE = .TRUE.
    END IF
*
    IF( ITYPE.EQ.0 ) THEN
*
*       Full matrix
*
        DO 30 J = 1, N
            DO 20 I = 1, M
                A( I, J ) = A( I, J )*MUL
20          CONTINUE
30          CONTINUE
*
        ELSE IF( ITYPE.EQ.1 ) THEN
*
*       Lower triangular matrix
*
        DO 50 J = 1, N
            DO 40 I = J, M
                A( I, J ) = A( I, J )*MUL
40          CONTINUE
50          CONTINUE
*
        ELSE IF( ITYPE.EQ.2 ) THEN
*
*       Upper triangular matrix
*
        DO 70 J = 1, N
            DO 60 I = 1, MIN( J, M )
                A( I, J ) = A( I, J )*MUL
60          CONTINUE
70          CONTINUE
*
        ELSE IF( ITYPE.EQ.3 ) THEN

```



```

*
*      Upper Hessenberg matrix
*
      DO 90 J = 1, N
        DO 80 I = 1, MIN( J+1, M )
          A( I, J ) = A( I, J )*MUL
80      CONTINUE
90      CONTINUE
*
      ELSE IF( ITYPE.EQ.4 ) THEN
*
*      Lower half of a symmetric band matrix
*
      K3 = KL + 1
      K4 = N + 1
      DO 110 J = 1, N
        DO 100 I = 1, MIN( K3, K4-J )
          A( I, J ) = A( I, J )*MUL
100      CONTINUE
110      CONTINUE
*
      ELSE IF( ITYPE.EQ.5 ) THEN
*
*      Upper half of a symmetric band matrix
*
      K1 = KU + 2
      K3 = KU + 1
      DO 130 J = 1, N
        DO 120 I = MAX( K1-J, 1 ), K3
          A( I, J ) = A( I, J )*MUL
120      CONTINUE
130      CONTINUE
*
      ELSE IF( ITYPE.EQ.6 ) THEN
*
*      Band matrix
*
      K1 = KL + KU + 2
      K2 = KL + 1
      K3 = 2*KL + KU + 1
      K4 = KL + KU + 1 + M
      DO 150 J = 1, N
        DO 140 I = MAX( K1-J, K2 ), MIN( K3, K4-J )
          A( I, J ) = A( I, J )*MUL
140      CONTINUE
150      CONTINUE
*
      END IF
*
      IF( .NOT.DONE )

```

```

$    GO TO 10
*
    RETURN
*
*    End of DLASCL
*
    END

```

— LAPACK dlascl —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dlascl (type kl ku cfrom cto m n a lda info)
    (declare (type (simple-array double-float (*)) a)
              (type (double-float) cto cfrom)
              (type fixnum info lda n m ku kl)
              (type character type))
    (f2cl-lib:with-multi-array-data
      ((type double-float type-%data% type-%offset%)
       (a double-float a-%data% a-%offset%))
      (prog ((bignum 0.0) (cfrom1 0.0) (cfromc 0.0) (cto1 0.0) (ctoc 0.0)
              (mul 0.0) (smlnum 0.0) (i 0) (itype 0) (j 0) (k1 0) (k2 0) (k3 0)
              (k4 0) (done nil))
        (declare (type (double-float) bignum cfrom1 cfromc cto1 ctoc mul
                        smlnum)
                  (type fixnum i itype j k1 k2 k3 k4)
                  (type (member t nil) done))
        (setf info 0)
        (cond
          ((char-equal type #\G)
           (setf itype 0))
          ((char-equal type #\L)
           (setf itype 1))
          ((char-equal type #\U)
           (setf itype 2))
          ((char-equal type #\H)
           (setf itype 3))
          ((char-equal type #\B)
           (setf itype 4))
          ((char-equal type #\Q)
           (setf itype 5))
          ((char-equal type #\Z)
           (setf itype 6))
          (t
           (setf itype -1)))

```

```

(cond
  ((= itype (f2cl-lib:int-sub 1))
    (setf info -1))
  ((= cfrom zero)
    (setf info -4))
  ((< m 0)
    (setf info -6))
  ((or (< n 0) (and (= itype 4) (/= n m)) (and (= itype 5) (/= n m)))
    (setf info -7))
  ((and (<= itype 3)
        (< lda
          (max (the fixnum 1) (the fixnum m))))
    (setf info -9))
  ((>= itype 4)
    (cond
      ((or (< kl 0)
            (> kl
              (max
                (the fixnum
                  (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                (the fixnum 0))))
        (setf info -2))
      ((or (< ku 0)
            (> ku
              (max
                (the fixnum
                  (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
                (the fixnum 0))))
        (and (or (= itype 4) (= itype 5)) (/= kl ku)))
        (setf info -3))
      ((or (and (= itype 4) (< lda (f2cl-lib:int-add kl 1)))
            (and (= itype 5) (< lda (f2cl-lib:int-add ku 1)))
            (and (= itype 6)
              (< lda (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))))
        (setf info -9))))))
    (cond
      ((/= info 0)
        (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DLASCL" (f2cl-lib:int-sub info))
        (go end_label)))
      (if (or (= n 0) (= m 0)) (go end_label))
      (setf smlnum (dlamch "S"))
      (setf bignum (/ one smlnum))
      (setf cfromc cfrom)
      (setf ctoc cto)
label10
      (setf cfrom1 (* cfromc smlnum))
      (setf cto1 (/ ctoc bignum))
      (cond

```

```

((and (> (abs cfrom1) (abs ctoc)) (/= ctoc zero))
  (setf mul smlnum)
  (setf done nil)
  (setf cfromc cfrom1))
(> (abs ctoc) (abs cfromc))
  (setf mul bignum)
  (setf done nil)
  (setf ctoc ctoc1))
(t
  (setf mul (/ ctoc cfromc))
  (setf done t)))
(cond
  ((= itype 0)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
              (*
                (f2cl-lib:fref a-%data%
                                (i j)
                                ((1 lda) (1 *))
                                a-%offset%)
                mul))))))))
    ((= itype 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                                  (i j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                (*
                  (f2cl-lib:fref a-%data%
                                  (i j)
                                  ((1 lda) (1 *))
                                  a-%offset%)
                  mul))))))))
    ((= itype 2)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody

```

```

(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i
    (min (the fixnum j)
          (the fixnum m)))
    nil)
(tagbody
  (setf (f2cl-lib:fref a-%data%
                     (i j)
                     ((1 lda) (1 *))
                     a-%offset%)
    (*
      (f2cl-lib:fref a-%data%
                     (i j)
                     ((1 lda) (1 *))
                     a-%offset%)
      mul))))))
(= itype 3)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i
      (min
        (the fixnum
          (f2cl-lib:int-add j 1))
        (the fixnum m)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
                           (i j)
                           ((1 lda) (1 *))
                           a-%offset%)
        (*
          (f2cl-lib:fref a-%data%
                           (i j)
                           ((1 lda) (1 *))
                           a-%offset%)
          mul))))))
(= itype 4)
(setf k3 (f2cl-lib:int-add k1 1))
(setf k4 (f2cl-lib:int-add n 1))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i
      (min (the fixnum k3)
            (the fixnum
              (f2cl-lib:int-add k4
                (f2cl-lib:int-sub

```

```

j))))))
      nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
        (*
          (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
          mul))))))
  (>= itype 5)
  (setf k1 (f2cl-lib:int-add ku 2))
  (setf k3 (f2cl-lib:int-add ku 1))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i
                  (max
                    (the fixnum
                     (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
                    (the fixnum 1))
                  (f2cl-lib:int-add i 1))
      (> i k3) nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
        (*
          (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
          mul))))))
  (>= itype 6)
  (setf k1 (f2cl-lib:int-add kl ku 2))
  (setf k2 (f2cl-lib:int-add kl 1))
  (setf k3 (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))
  (setf k4 (f2cl-lib:int-add kl ku 1 m))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i
                  (max
                    (the fixnum
                     (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
                    (the fixnum k2)))

```

```

                                (f2cl-lib:int-add i 1))
                                (> i
                                 (min (the fixnum k3)
                                      (the fixnum
                                       (f2cl-lib:int-add k4
                                                           (f2cl-lib:int-sub
                                                            j))))))
                                nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%)
            (*
              (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *))
                              a-%offset%)
              (mul))))))
    (if (not done) (go label10))
  end_label
  (return (values nil nil nil nil nil nil nil nil info))))

```

dlasd0 LAPACK

— dlasd0.input —

```

)set break resume
)sys rm -f dlasd0.output
)spool dlasd0.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd0.help —

```

=====
dlasd0 examples

```

```
=====
=====
Man Page Details
=====
```

NAME

DLASD0 - divide and conquer approach, DLASD0 computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where $M = N + \text{SQRE}$

SYNOPSIS

```
SUBROUTINE DLASD0( N, SQRE, D, E, U, LDU, VT, LDVT, SMLSIZ, IWORK,
                  WORK, INFO )
```

```
      INTEGER      INFO, LDU, LDVT, N, SMLSIZ, SQRE
```

```
      INTEGER      IWORK( * )
```

```
      DOUBLE      PRECISION D( * ), E( * ), U( LDU, * ), VT( LDVT, *
                  ), WORK( * )
```

Purpose

```
=====
```

Using a divide and conquer approach, DLASD0 computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where $M = N + \text{SQRE}$. The algorithm computes orthogonal matrices U and VT such that $B = U * S * VT$. The singular values S are overwritten on D.

A related subroutine, DLASDA, computes only the singular values, and optionally, the singular vectors in compact form.

Arguments

```
=====
```

N (input) INTEGER
On entry, the row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D.

SQRE (input) INTEGER
Specifies the column dimension of the bidiagonal matrix.
= 0: The bidiagonal matrix has column dimension $M = N$;
= 1: The bidiagonal matrix has column dimension $M = N+1$;

D (input/output) DOUBLE PRECISION array, dimension (N)
On entry D contains the main diagonal of the bidiagonal matrix.
On exit D, if $\text{INFO} = 0$, contains its singular values.

E (input) DOUBLE PRECISION array, dimension (M-1)
 Contains the subdiagonal entries of the bidiagonal matrix.
 On exit, E has been destroyed.

U (output) DOUBLE PRECISION array, dimension at least (LDQ, N)
 On exit, U contains the left singular vectors.

LDU (input) INTEGER
 On entry, leading dimension of U.

VT (output) DOUBLE PRECISION array, dimension at least (LDVT, M)
 On exit, VT' contains the right singular vectors.

LDVT (input) INTEGER
 On entry, leading dimension of VT.

SMLSIZ (input) INTEGER
 On entry, maximum size of the subproblems at the
 bottom of the computation tree.

IWORK INTEGER work array.
 Dimension must be at least (8 * N)

WORK DOUBLE PRECISION work array.
 Dimension must be at least (3 * M**2 + 2 * M)

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: if INFO = 1, an singular value did not converge

Further Details
 =====

Based on contributions by
 Ming Gu and Huan Ren, Computer Science Division, University of
 California at Berkeley, USA

— dlasd0.f —

```

SUBROUTINE DLASDO( N, SQRE, D, E, U, LDU, VT, LDVT, SMLSIZ, IWORK,
$                WORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
```

```

*      Courant Institute, Argonne National Lab, and Rice University
*      June 30, 1999
*
*      .. Scalar Arguments ..
      INTEGER          INFO, LDU, LDVT, N, SMLSIZ, SQRE
*
*      ..
*      .. Array Arguments ..
      INTEGER          IWORK( * )
      DOUBLE PRECISION D( * ), E( * ), U( LDU, * ), VT( LDVT, * ),
$                     WORK( * )
*
*      ..
*
*      =====
*
*      .. Local Scalars ..
      INTEGER          I, I1, IC, IDXQ, IDXQC, IM1, INODE, ITEMP, IWK,
$                     J, LF, LL, LVL, M, NCC, ND, NDB1, NDIML, NDIMR,
$                     NL, NLF, NLP1, NLVL, NR, NRF, NRP1, SQREI
      DOUBLE PRECISION ALPHA, BETA
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DLASD1, DLASDQ, DLASDT, XERBLA
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
*      INFO = 0
*
*      IF( N.LT.0 ) THEN
*         INFO = -1
*      ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN
*         INFO = -2
*      END IF
*
*      M = N + SQRE
*
*      IF( LDU.LT.N ) THEN
*         INFO = -6
*      ELSE IF( LDVT.LT.M ) THEN
*         INFO = -8
*      ELSE IF( SMLSIZ.LT.3 ) THEN
*         INFO = -9
*      END IF
*      IF( INFO.NE.0 ) THEN
*         CALL XERBLA( 'DLASDQ', -INFO )
*         RETURN
*      END IF
*
*      If the input matrix is too small, call DLASDQ to find the SVD.

```

```

*
      IF( N.LE.SMLSIZ ) THEN
        CALL DLASDQ( 'U', SQRE, N, M, N, 0, D, E, VT, LDVT, U, LDU, U,
$              LDU, WORK, INFO )
        RETURN
      END IF
*
*      Set up the computation tree.
*
      INODE = 1
      NDIML = INODE + N
      NDIMR = NDIML + N
      IDXQ = NDIMR + N
      IWK = IDXQ + N
      CALL DLASDT( N, NLVL, ND, IWORK( INODE ), IWORK( NDIML ),
$              IWORK( NDIMR ), SMLSIZ )
*
*      For the nodes on bottom level of the tree, solve
*      their subproblems by DLASDQ.
*
      NDB1 = ( ND+1 ) / 2
      NCC = 0
      DO 30 I = NDB1, ND
*
*      IC : center row of each node
*      NL : number of rows of left  subproblem
*      NR : number of rows of right subproblem
*      NLF: starting row of the left  subproblem
*      NRF: starting row of the right subproblem
*
        I1 = I - 1
        IC = IWORK( INODE+I1 )
        NL = IWORK( NDIML+I1 )
        NLP1 = NL + 1
        NR = IWORK( NDIMR+I1 )
        NRP1 = NR + 1
        NLF = IC - NL
        NRF = IC + 1
        SQREI = 1
        CALL DLASDQ( 'U', SQREI, NL, NLP1, NL, NCC, D( NLF ), E( NLF ),
$              VT( NLF, NLF ), LDVT, U( NLF, NLF ), LDU,
$              U( NLF, NLF ), LDU, WORK, INFO )
        IF( INFO.NE.0 ) THEN
          RETURN
        END IF
        ITEMP = IDXQ + NLF - 2
        DO 10 J = 1, NL
          IWORK( ITEMP+J ) = J
10      CONTINUE
        IF( I.EQ.ND ) THEN

```

```

        SQREI = SQRE
    ELSE
        SQREI = 1
    END IF
    NRP1 = NR + SQREI
    CALL DLASDQ( 'U', SQREI, NR, NRP1, NR, NCC, D( NRF ), E( NRF ),
$           VT( NRF, NRF ), LDVT, U( NRF, NRF ), LDU,
$           U( NRF, NRF ), LDU, WORK, INFO )
    IF( INFO.NE.0 ) THEN
        RETURN
    END IF
    ITEMP = IDXQ + IC
    DO 20 J = 1, NR
        IWORK( ITEMP+J-1 ) = J
20    CONTINUE
30 CONTINUE
*
*   Now conquer each subproblem bottom-up.
*
    DO 50 LVL = NLVL, 1, -1
*
*   Find the first node LF and last node LL on the
*   current level LVL.
*
        IF( LVL.EQ.1 ) THEN
            LF = 1
            LL = 1
        ELSE
            LF = 2**( LVL-1 )
            LL = 2*LF - 1
        END IF
        DO 40 I = LF, LL
            IM1 = I - 1
            IC = IWORK( INODE+IM1 )
            NL = IWORK( NDIML+IM1 )
            NR = IWORK( NDIRM+IM1 )
            NLF = IC - NL
            IF( ( SQRE.EQ.0 ) .AND. ( I.EQ.LL ) ) THEN
                SQREI = SQRE
            ELSE
                SQREI = 1
            END IF
            IDXQC = IDXQ + NLF - 1
            ALPHA = D( IC )
            BETA = E( IC )
            CALL DLASD1( NL, NR, SQREI, D( NLF ), ALPHA, BETA,
$                   U( NLF, NLF ), LDU, VT( NLF, NLF ), LDVT,
$                   IWORK( IDXQC ), IWORK( IWK ), WORK, INFO )
            IF( INFO.NE.0 ) THEN
                RETURN
            
```

```

        END IF
40    CONTINUE
50 CONTINUE
*
    RETURN
*
*    End of DLASD0
*
    END

```

— LAPACK dlasd0 —

```

(defun dlasd0 (n sqre d e u ldu vt ldvt smlsiz iwork work info)
  (declare (type (simple-array fixnum (*)) iwork)
            (type (simple-array double-float (*)) work vt u e d)
            (type fixnum info smlsiz ldvt ldu sqre n))
  (f2cl-lib:with-multi-array-data
    ((d double-float d-%data% d-%offset%)
     (e double-float e-%data% e-%offset%)
     (u double-float u-%data% u-%offset%)
     (vt double-float vt-%data% vt-%offset%)
     (work double-float work-%data% work-%offset%)
     (iwork fixnum iwork-%data% iwork-%offset%))
    (prog ((alpha 0.0) (beta 0.0) (i 0) (i1 0) (ic 0) (idxq 0) (idxqc 0)
           (im1 0) (inode 0) (itemp 0) (iwk 0) (j 0) (lf 0) (ll 0) (lvl 0)
           (m 0) (ncc 0) (nd 0) (ndb1 0) (ndiml 0) (ndimr 0) (nl 0) (nlf 0)
           (nlp1 0) (nlvl 0) (nr 0) (nrf 0) (nrp1 0) (sqrei 0))
      (declare (type fixnum sqrei nrp1 nrf nr nlvl nlp1 nlf nl
                                ndimr ndiml ndb1 nd ncc m lvl ll lf j
                                iwkw itemp inode im1 idxqc idxq ic i1
                                i)
                (type (double-float) beta alpha))
      (setf info 0)
      (cond
        (((< n 0)
          (setf info -1))
         ((or (< sqre 0) (> sqre 1))
          (setf info -2)))
        (setf m (f2cl-lib:int-add n sqre))
        (cond
          (((< ldu n)
            (setf info -6))
           ((< ldvt m)
            (setf info -8))
           ((< smlsiz 3)
            (setf info -9)))

```

```

(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DLASD0" (f2cl-lib:int-sub info))
    (go end_label)))
(cond
  ((<= n smlsiz)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12 var-13 var-14 var-15)
      (dlasdq "U" sqre n m n 0 d e vt ldvt u ldu u ldu work info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                       var-8 var-9 var-10 var-11 var-12 var-13 var-14))
      (setf info var-15))
    (go end_label)))
(setf inode 1)
(setf ndiml (f2cl-lib:int-add inode n))
(setf ndimr (f2cl-lib:int-add ndiml n))
(setf idxq (f2cl-lib:int-add ndimr n))
(setf iwk (f2cl-lib:int-add idxq n))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (dlasdt n nlvl nd
    (f2cl-lib:array-slice iwork fixnum (inode) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (ndiml) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (ndimr) ((1 *)))
    smlsiz)
  (declare (ignore var-0 var-3 var-4 var-5 var-6))
  (setf nlvl var-1)
  (setf nd var-2))
(setf ndb1 (the fixnum (truncate (+ nd 1) 2)))
(setf ncc 0)
(f2cl-lib:fdo (i ndb1 (f2cl-lib:int-add i 1))
  (> i nd) nil)
(tagbody
  (setf i1 (f2cl-lib:int-sub i 1))
  (setf ic
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add inode i1))
      ((1 *))
      iwork-%offset%))
  (setf nl
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add ndiml i1))
      ((1 *))
      iwork-%offset%))
  (setf nlp1 (f2cl-lib:int-add nl 1))
  (setf nr
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add ndimr i1))

```

```

                                ((1 *))
                                iwork-%offset%))
(setf nrp1 (f2cl-lib:int-add nr 1))
(setf nlf (f2cl-lib:int-sub ic nl))
(setf nrf (f2cl-lib:int-add ic 1))
(setf sqrei 1)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11 var-12 var-13 var-14 var-15)
  (dlasdq "U" sqrei nl nlpi nl ncc
   (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
   (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
   (f2cl-lib:array-slice vt
    double-float
    (nlf nlf)
    ((1 ldvt) (1 *)))

    ldvt
    (f2cl-lib:array-slice u double-float (nlf nlf) ((1 ldu) (1 *)))
    ldu
    (f2cl-lib:array-slice u double-float (nlf nlf) ((1 ldu) (1 *)))
    ldu work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10 var-11 var-12 var-13 var-14))
  (setf info var-15))
(cond
  ((/= info 0)
   (go end_label)))
(setf itemp (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 2))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j nl) nil)
  (tagbody
    (setf (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add itemp j))
      ((1 *))
      iwork-%offset%)
      j)))
(cond
  ((= i nd)
   (setf sqrei sqre))
  (t
   (setf sqrei 1)))
(setf nrp1 (f2cl-lib:int-add nr sqrei))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11 var-12 var-13 var-14 var-15)
  (dlasdq "U" sqrei nr nrp1 nr ncc
   (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
   (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
   (f2cl-lib:array-slice vt
    double-float

```

```

(nrf nrf)
((1 ldvt) (1 *)))

ldvt
(f2cl-lib:array-slice u double-float (nrf nrf) ((1 ldu) (1 *)))
ldu
(f2cl-lib:array-slice u double-float (nrf nrf) ((1 ldu) (1 *)))
ldu work info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
               var-8 var-9 var-10 var-11 var-12 var-13 var-14))
(setf info var-15))
(cond
  ((/= info 0)
   (go end_label)))
(setf itemp (f2cl-lib:int-add idxq ic))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j nr) nil)
(tagbody
  (setf (f2cl-lib:fref iwork-%data%
                      ((f2cl-lib:int-sub
                        (f2cl-lib:int-add itemp j)
                        1))
                      ((1 *))
                      iwork-%offset%
                      j))))
(f2cl-lib:fdo (lvl nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1)))
  (> lvl 1) nil)
(tagbody
  (cond
    ((= lvl 1)
     (setf lf 1)
     (setf ll 1))
    (t
     (setf lf (expt 2 (f2cl-lib:int-sub lvl 1)))
     (setf ll (f2cl-lib:int-sub (f2cl-lib:int-mul 2 lf) 1))))
  (f2cl-lib:fdo (i lf (f2cl-lib:int-add i 1))
    (> i ll) nil)
  (tagbody
    (setf im1 (f2cl-lib:int-sub i 1))
    (setf ic
      (f2cl-lib:fref iwork-%data%
                      ((f2cl-lib:int-add inode im1)
                      ((1 *))
                      iwork-%offset%)))
    (setf nl
      (f2cl-lib:fref iwork-%data%
                      ((f2cl-lib:int-add ndiml im1)
                      ((1 *))
                      iwork-%offset%)))
    (setf nr
      (f2cl-lib:fref iwork-%data%

```



```

                                ((f2cl-lib:int-add ndimr im1))
                                ((1 *))
                                iwork-%offset%))
(setf nlf (f2cl-lib:int-sub ic nlf))
(cond
  ((and (= sqre 0) (= i 11))
    (setf sqrei sqre))
  (t
    (setf sqrei 1)))
(setf idxqc (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 1))
(setf alpha (f2cl-lib:fref d-%data% (ic) ((1 *)) d-%offset%))
(setf beta (f2cl-lib:fref e-%data% (ic) ((1 *)) e-%offset%))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13)
  (dlasd1 nlf nr sqrei
    (f2cl-lib:array-slice d double-float (nlf) ((1 *))) alpha
    beta
    (f2cl-lib:array-slice u
      double-float
      (nlf nlf)
      ((1 ldu) (1 *)))
    ldu
    (f2cl-lib:array-slice vt
      double-float
      (nlf nlf)
      ((1 ldvt) (1 *)))
    ldvt
    (f2cl-lib:array-slice iwork
      fixnum
      (idxqc)
      ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (iwk) ((1 *)))
    work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-6 var-7 var-8
    var-9 var-10 var-11 var-12))
  (setf alpha var-4)
  (setf beta var-5)
  (setf info var-13))
(cond
  ((/= info 0)
    (go end_label))))))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil info))))

```

dlasd1 LAPACK**— dlasd1.input —**

```

)set break resume
)sys rm -f dlasd1.output
)spool dlasd1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd1.help —

```

=====
dlasd1 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASD1 - the SVD of an upper bidiagonal N-by-M matrix B,

SYNOPSIS

```

SUBROUTINE DLASD1( NL, NR, SQRE, D, ALPHA, BETA, U, LDU, VT, LDVT,
                  IDXQ, IWORK, WORK, INFO )

```

INTEGER INFO, LDU, LDVT, NL, NR, SQRE

DOUBLE PRECISION ALPHA, BETA

INTEGER IDXQ(*), IWORK(*)

DOUBLE PRECISION D(*), U(LDU, *), VT(LDVT, *), WORK(*)

Purpose

```

=====

```

DLASD1 computes the SVD of an upper bidiagonal N-by-M matrix B, where $N = NL + NR + 1$ and $M = N + SQRE$. DLASD1 is called from DLASD0.

A related subroutine DLASD7 handles the case in which the singular values (and the singular vectors in factored form) are desired.

DLASD1 computes the SVD as follows:

$$\begin{aligned}
 B &= U(\text{in}) * \begin{pmatrix} D1(\text{in}) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(\text{in}) & 0 \end{pmatrix} * VT(\text{in}) \\
 &= U(\text{out}) * \begin{pmatrix} D(\text{out}) & 0 \end{pmatrix} * VT(\text{out})
 \end{aligned}$$

where $Z' = (Z1' \ a \ Z2' \ b) = u' \ VT'$, and u is a vector of dimension M with ALPHA and BETA in the $NL+1$ and $NL+2$ th entries and zeros elsewhere; and the entry b is empty if $SQRE = 0$.

The left singular vectors of the original matrix are stored in U , and the transpose of the right singular vectors are stored in VT , and the singular values are in D . The algorithm consists of three stages:

The first stage consists of deflating the size of the problem when there are multiple singular values or when there are zeros in the Z vector. For each such occurrence the dimension of the secular equation problem is reduced by one. This stage is performed by the routine DLASD2.

The second stage consists of calculating the updated singular values. This is done by finding the square roots of the roots of the secular equation via the routine DLASD4 (as called by DLASD3). This routine also calculates the singular vectors of the current problem.

The final stage consists of computing the updated singular vectors directly using the updated singular values. The singular vectors for the current problem are multiplied with the singular vectors from the overall problem.

Arguments

=====

NL (input) INTEGER
The row dimension of the upper block. $NL \geq 1$.

NR (input) INTEGER
The row dimension of the lower block. $NR \geq 1$.

SQRE (input) INTEGER
= 0: the lower block is an NR -by- NR square matrix.
= 1: the lower block is an NR -by- $(NR+1)$ rectangular matrix.

The bidiagonal matrix has row dimension $N = NL + NR + 1$,

and column dimension $M = N + \text{SQRE}$.

- D** (input/output) DOUBLE PRECISION array,
dimension $(N = \text{NL} + \text{NR} + 1)$.
On entry $D(1:\text{NL}, 1:\text{NL})$ contains the singular values of the upper block; and $D(\text{NL} + 2:N)$ contains the singular values of the lower block. On exit $D(1:N)$ contains the singular values of the modified matrix.
- ALPHA** (input) DOUBLE PRECISION
Contains the diagonal element associated with the added row.
- BETA** (input) DOUBLE PRECISION
Contains the off-diagonal element associated with the added row.
- U** (input/output) DOUBLE PRECISION array, dimension (LDU, N)
On entry $U(1:\text{NL}, 1:\text{NL})$ contains the left singular vectors of the upper block; $U(\text{NL} + 2:N, \text{NL} + 2:N)$ contains the left singular vectors of the lower block. On exit U contains the left singular vectors of the bidiagonal matrix.
- LDU** (input) INTEGER
The leading dimension of the array U . $\text{LDU} \geq \max(1, N)$.
- VT** (input/output) DOUBLE PRECISION array, dimension (LDVT, M)
where $M = N + \text{SQRE}$.
On entry $VT(1:\text{NL} + 1, 1:\text{NL} + 1)$ contains the right singular vectors of the upper block; $VT(\text{NL} + 2:M, \text{NL} + 2:M)$ contains the right singular vectors of the lower block. On exit VT contains the right singular vectors of the bidiagonal matrix.
- LDVT** (input) INTEGER
The leading dimension of the array VT . $\text{LDVT} \geq \max(1, M)$.
- IDXQ** (output) INTEGER array, dimension (N)
This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e.
 $D(\text{IDXQ}(I = 1, N))$ will be in ascending order.
- IWORK** (workspace) INTEGER array, dimension $(4 * N)$
- WORK** (workspace) DOUBLE PRECISION array, dimension $(3 * M^2 + 2 * M)$
- INFO** (output) INTEGER
= 0: successful exit.
< 0: if $\text{INFO} = -i$, the i -th argument had an illegal value.
> 0: if $\text{INFO} = 1$, a singular value did not converge

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of
California at Berkeley, USA

— dlasd1.f —

```

SUBROUTINE DLASD1( NL, NR, SQRE, D, ALPHA, BETA, U, LDU, VT, LDVT,
$                IDXQ, IWORK, WORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      INTEGER          INFO, LDU, LDVT, NL, NR, SQRE
      DOUBLE PRECISION ALPHA, BETA
*
*  .. Array Arguments ..
      INTEGER          IDXQ( * ), IWORK( * )
      DOUBLE PRECISION D( * ), U( LDU, * ), VT( LDVT, * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
*
      DOUBLE PRECISION ONE, ZERO
      PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*  .. Local Scalars ..
      INTEGER          COLTYP, I, IDX, IDXC, IDXP, IQ, ISIGMA, IU2,
$                    IVT2, IZ, K, LDQ, LDU2, LDVT2, M, N, N1, N2
      DOUBLE PRECISION ORGNRM
*
*  .. External Subroutines ..
      EXTERNAL         DLAMRG, DLASCL, DLASD2, DLASD3, XERBLA
*
*  .. Intrinsic Functions ..
      INTRINSIC        ABS, MAX
*
*  .. Executable Statements ..
*

```

```

*      Test the input parameters.
*
      INFO = 0
*
      IF( NL.LT.1 ) THEN
          INFO = -1
      ELSE IF( NR.LT.1 ) THEN
          INFO = -2
      ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN
          INFO = -3
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DLASD1', -INFO )
          RETURN
      END IF
*
      N = NL + NR + 1
      M = N + SQRE
*
*      The following values are for bookkeeping purposes only. They are
*      integer pointers which indicate the portion of the workspace
*      used by a particular array in DLASD2 and DLASD3.
*
      LDU2 = N
      LDVT2 = M
*
      IZ = 1
      ISIGMA = IZ + M
      IU2 = ISIGMA + N
      IVT2 = IU2 + LDU2*N
      IQ = IVT2 + LDVT2*M
*
      IDX = 1
      IDXC = IDX + N
      COLTYP = IDXC + N
      IDXP = COLTYP + N
*
*      Scale.
*
      ORGNRM = MAX( ABS( ALPHA ), ABS( BETA ) )
      D( NL+1 ) = ZERO
      DO 10 I = 1, N
          IF( ABS( D( I ) ) .GT. ORGNRM ) THEN
              ORGNRM = ABS( D( I ) )
          END IF
10 CONTINUE
      CALL DLASCL( 'G', 0, 0, ORGNRM, ONE, N, 1, D, N, INFO )
      ALPHA = ALPHA / ORGNRM
      BETA = BETA / ORGNRM
*

```

```

*      Deflate singular values.
*
      CALL DLASD2( NL, NR, SQRE, K, D, WORK( IZ ), ALPHA, BETA, U, LDU,
$              VT, LDVT, WORK( ISIGMA ), WORK( IU2 ), LDU2,
$              WORK( IVT2 ), LDVT2, IWORK( IDXP ), IWORK( IDX ),
$              IWORK( IDXC ), IDXQ, IWORK( COLTYP ), INFO )
*
*      Solve Secular Equation and update singular vectors.
*
      LDQ = K
      CALL DLASD3( NL, NR, SQRE, K, D, WORK( IQ ), LDQ, WORK( ISIGMA ),
$              U, LDU, WORK( IU2 ), LDU2, VT, LDVT, WORK( IVT2 ),
$              LDVT2, IWORK( IDXC ), IWORK( COLTYP ), WORK( IZ ),
$              INFO )
      IF( INFO.NE.0 ) THEN
        RETURN
      END IF
*
*      Unscale.
*
      CALL DLASCL( 'G', 0, 0, ONE, ORGNRM, N, 1, D, N, INFO )
*
*      Prepare the IDXQ sorting permutation.
*
      N1 = K
      N2 = N - K
      CALL DLAMRG( N1, N2, D, 1, -1, IDXQ )
*
      RETURN
*
*      End of DLASD1
*
      END

```

— LAPACK dlasd1 —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dlasd1 (nl nr sqre d alpha beta u ldu vt ldvt idxq iwork work info)
    (declare (type (simple-array fixnum (*)) iwork idxq)
              (type (double-float) beta alpha)
              (type (simple-array double-float (*)) work vt u d)
              (type fixnum info ldvt ldu sqre nr nl))
    (f2cl-lib:with-multi-array-data
      ((d double-float d-%data% d-%offset%)

```

```

(u double-float u-%data% u-%offset%)
(vt double-float vt-%data% vt-%offset%)
(work double-float work-%data% work-%offset%)
(idxq fixnum idxq-%data% idxq-%offset%)
(iwork fixnum iwork-%data% iwork-%offset%))
(prog ((orgnrm 0.0) (coltyp 0) (i 0) (idx 0) (idxc 0) (idxp 0) (iq 0)
      (isigma 0) (iu2 0) (ivt2 0) (iz 0) (k 0) (ldq 0) (ldu2 0)
      (ldvt2 0) (m 0) (n 0) (n1 0) (n2 0))
(declare (type (double-float) orgnrm)
          (type fixnum coltyp i idx idxc idxp iq isigma iu2
                                ivt2 iz k ldq ldu2 ldvt2 m n n1 n2))

(setf info 0)
(cond
  ((< n1 1)
   (setf info -1))
  ((< nr 1)
   (setf info -2))
  ((or (< sqre 0) (> sqre 1))
   (setf info -3)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD1" (f2cl-lib:int-sub info))
   (go end_label)))
(setf n (f2cl-lib:int-add n1 nr 1))
(setf m (f2cl-lib:int-add n sqre))
(setf ldu2 n)
(setf ldvt2 m)
(setf iz 1)
(setf isigma (f2cl-lib:int-add iz m))
(setf iu2 (f2cl-lib:int-add isigma n))
(setf ivt2 (f2cl-lib:int-add iu2 (f2cl-lib:int-mul ldu2 n)))
(setf iq (f2cl-lib:int-add ivt2 (f2cl-lib:int-mul ldvt2 m)))
(setf idx 1)
(setf idxc (f2cl-lib:int-add idx n))
(setf coltyp (f2cl-lib:int-add idxc n))
(setf idxp (f2cl-lib:int-add coltyp n))
(setf orgnrm (max (abs alpha) (abs beta)))
(setf (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add n1 1))
                    ((1 *))
                    d-%offset%)
      zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
  (tagbody
   (cond
    ((> (abs (f2cl-lib:fref d (i) ((1 *)))) orgnrm)
     (setf orgnrm

```



```

        (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 orgnrm one n 1 d n info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf info var-9))
(setf alpha (/ alpha orgnrm))
(setf beta (/ beta orgnrm))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
    var-19 var-20 var-21 var-22)
  (dlasd2 nl nr sqre k d
    (f2cl-lib:array-slice work double-float (iz) ((1 *))) alpha beta u
    ldu vt ldvt
    (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
    (f2cl-lib:array-slice work double-float (iu2) ((1 *))) ldu2
    (f2cl-lib:array-slice work double-float (ivt2) ((1 *))) ldvt2
    (f2cl-lib:array-slice iwork fixnum (idxp) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (idx) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (idxc) ((1 *))) idxq
    (f2cl-lib:array-slice iwork fixnum (coltyp) ((1 *)))
    info)
  (declare (ignore var-0 var-1 var-2 var-4 var-5 var-6 var-7 var-8
    var-9 var-10 var-11 var-12 var-13 var-14 var-15
    var-16 var-17 var-18 var-19 var-20 var-21))
  (setf k var-3)
  (setf info var-22))
(setf ldq k)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
    var-19)
  (dlasd3 nl nr sqre k d
    (f2cl-lib:array-slice work double-float (iq) ((1 *))) ldq
    (f2cl-lib:array-slice work double-float (isigma) ((1 *))) u ldu
    (f2cl-lib:array-slice work double-float (iu2) ((1 *))) ldu2 vt
    ldvt (f2cl-lib:array-slice work double-float (ivt2) ((1 *))) ldvt2
    (f2cl-lib:array-slice iwork fixnum (idxc) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (coltyp) ((1 *)))
    (f2cl-lib:array-slice work double-float (iz) ((1 *))) info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10 var-11 var-12 var-13 var-14
    var-15 var-16 var-17 var-18))
  (setf info var-19))
(cond
  ((/= info 0)
   (go end_label)))
(multiple-value-bind

```

```

      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 one orgnrm n 1 d n info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8))
      (setf info var-9))
      (setf n1 k)
      (setf n2 (f2cl-lib:int-sub n k))
      (dlamrg n1 n2 d 1 -1 idxq)
end_label
      (return
      (values nil
              nil
              nil
              nil
              alpha
              beta
              nil
              nil
              nil
              nil
              nil
              nil
              nil
              nil
              info))))))

```

dlasd2 LAPACK

— dlasd2.input —

```

)set break resume
)sys rm -f dlasd2.output
)spool dlasd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd2.help —

=====

dlasd2 examples

=====

=====

Man Page Details

=====

NAME

DLASD2 - the two sets of singular values together into a single sorted set

SYNOPSIS

SUBROUTINE DLASD2(NL, NR, SQRE, K, D, Z, ALPHA, BETA, U, LDU, VT, LDVT, DSIGMA, U2, LDU2, VT2, LDVT2, IDXP, IDX, IDXC, IDXQ, COLTYP, INFO)

INTEGER INFO, K, LDU, LDU2, LDVT, LDVT2, NL, NR, SQRE

DOUBLE PRECISION ALPHA, BETA

INTEGER COLTYP(*), IDX(*), IDXC(*), IDXP(*), IDXQ(*)

DOUBLE PRECISION D(*), DSIGMA(*), U(LDU, *), U2(LDU2, *), VT(LDVT, *), VT2(LDVT2, *), Z(*)

Purpose

=====

DLASD2 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

DLASD2 is called from DLASD1.

Arguments

=====

NL (input) INTEGER
The row dimension of the upper block. NL >= 1.

NR (input) INTEGER
The row dimension of the lower block. NR >= 1.

SQRE (input) INTEGER
= 0: the lower block is an NR-by-NR square matrix.
= 1: the lower block is an NR-by-(NR+1) rectangular matrix.

The bidiagonal matrix has $N = NL + NR + 1$ rows and
 $M = N + SQRE \geq N$ columns.

- K** (output) INTEGER
 Contains the dimension of the non-deflated matrix,
 This is the order of the related secular equation. $1 \leq K \leq N$.
- D** (input/output) DOUBLE PRECISION array, dimension(N)
 On entry D contains the singular values of the two submatrices
 to be combined. On exit D contains the trailing (N-K) updated
 singular values (those which were deflated) sorted into
 increasing order.
- ALPHA** (input) DOUBLE PRECISION
 Contains the diagonal element associated with the added row.
- BETA** (input) DOUBLE PRECISION
 Contains the off-diagonal element associated with the added
 row.
- U** (input/output) DOUBLE PRECISION array, dimension(LDU,N)
 On entry U contains the left singular vectors of two
 submatrices in the two square blocks with corners at (1,1),
 (NL, NL), and (NL+2, NL+2), (N,N).
 On exit U contains the trailing (N-K) updated left singular
 vectors (those which were deflated) in its last N-K columns.
- LDU** (input) INTEGER
 The leading dimension of the array U. $LDU \geq N$.
- Z** (output) DOUBLE PRECISION array, dimension(N)
 On exit Z contains the updating row vector in the secular
 equation.
- DSIGMA** (output) DOUBLE PRECISION array, dimension (N)
 Contains a copy of the diagonal elements (K-1 singular values
 and one zero) in the secular equation.
- U2** (output) DOUBLE PRECISION array, dimension(LDU2,N)
 Contains a copy of the first K-1 left singular vectors which
 will be used by DLASD3 in a matrix multiply (DGEMM) to solve
 for the new left singular vectors. U2 is arranged into four
 blocks. The first block contains a column with 1 at NL+1 and
 zero everywhere else; the second block contains non-zero
 entries only at and above NL; the third contains non-zero
 entries only below NL+1; and the fourth is dense.
- LDU2** (input) INTEGER
 The leading dimension of the array U2. $LDU2 \geq N$.

- VT (input/output) DOUBLE PRECISION array, dimension(LDVT,M)
 On entry VT' contains the right singular vectors of two submatrices in the two square blocks with corners at (1,1), (NL+1, NL+1), and (NL+2, NL+2), (M,M).
 On exit VT' contains the trailing (N-K) updated right singular vectors (those which were deflated) in its last N-K columns. In case SQRE =1, the last row of VT spans the right null space.
- LDVT (input) INTEGER
 The leading dimension of the array VT. LDVT >= M.
- VT2 (output) DOUBLE PRECISION array, dimension(LDVT2,N)
 VT2' contains a copy of the first K right singular vectors which will be used by DLASD3 in a matrix multiply (DGEMM) to solve for the new right singular vectors. VT2 is arranged into three blocks. The first block contains a row that corresponds to the special 0 diagonal element in SIGMA; the second block contains non-zeros only at and before NL +1; the third block contains non-zeros only at and after NL +2.
- LDVT2 (input) INTEGER
 The leading dimension of the array VT2. LDVT2 >= M.
- IDXP (workspace) INTEGER array, dimension(N)
 This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.
- IDX (workspace) INTEGER array, dimension(N)
 This will contain the permutation used to sort the contents of D into ascending order.
- IDXC (output) INTEGER array, dimension(N)
 This will contain the permutation used to arrange the columns of the deflated U matrix into three groups: the first group contains non-zero entries only at and above NL, the second contains non-zero entries only below NL+2, and the third is dense.
- COLTYP (workspace/output) INTEGER array, dimension(N)
 As workspace, this will contain a label which will indicate which of the following types a column in the U2 matrix or a row in the VT2 matrix is:
 1 : non-zero in the upper half only
 2 : non-zero in the lower half only
 3 : dense
 4 : deflated

On exit, it is an array of dimension 4, with COLTYP(I) being the dimension of the I-th type columns.

IDXQ (input) INTEGER array, dimension(N)
This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first half of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = -i, the i-th argument had an illegal value.

Further Details
=====

Based on contributions by
Ming Gu and Huan Ren, Computer Science Division, University of
California at Berkeley, USA

— dlasd2.f —

```

      SUBROUTINE DLASD2( NL, NR, SQRE, K, D, Z, ALPHA, BETA, U, LDU, VT,
$                      LDVT, DSIGMA, U2, LDU2, VT2, LDVT2, IDXP, IDX,
$                      IDXC, IDXQ, COLTYP, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*  Courant Institute, NAG Ltd., and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
      INTEGER          INFO, K, LDU, LDU2, LDVT, LDVT2, NL, NR, SQRE
      DOUBLE PRECISION ALPHA, BETA
*
*  ..
*
*  .. Array Arguments ..
      INTEGER          COLTYP( * ), IDX( * ), IDXC( * ), IDXP( * ),
$                      IDXQ( * )
      DOUBLE PRECISION D( * ), DSIGMA( * ), U( LDU, * ),
$                      U2( LDU2, * ), VT( LDVT, * ), VT2( LDVT2, * ),
$                      Z( * )
*
*  ..
*
*  =====

```

```

*
*   .. Parameters ..
DOUBLE PRECISION    ZERO, ONE, TWO, EIGHT
PARAMETER           ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0,
$                   EIGHT = 8.0D+0 )
*
*   ..
*   .. Local Arrays ..
INTEGER             CTOT( 4 ), PSM( 4 )
*
*   ..
*   .. Local Scalars ..
INTEGER             CT, I, IDXI, IDXJ, IDXJP, J, JP, JPREV, K2, M,
$                   N, NLP1, NLP2
DOUBLE PRECISION    C, EPS, HLFTOL, S, TAU, TOL, Z1
*
*   ..
*   .. External Functions ..
DOUBLE PRECISION    DLAMCH, DLAPY2
EXTERNAL            DLAMCH, DLAPY2
*
*   ..
*   .. External Subroutines ..
EXTERNAL            DCOPY, DLACPY, DLAMRG, DLASET, DROT, XERBLA
*
*   ..
*   .. Intrinsic Functions ..
INTRINSIC            ABS, MAX
*
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
*   INFO = 0
*
*   IF( NL.LT.1 ) THEN
*       INFO = -1
*   ELSE IF( NR.LT.1 ) THEN
*       INFO = -2
*   ELSE IF( ( SQRE.NE.1 ) .AND. ( SQRE.NE.0 ) ) THEN
*       INFO = -3
*   END IF
*
*   N = NL + NR + 1
*   M = N + SQRE
*
*   IF( LDU.LT.N ) THEN
*       INFO = -10
*   ELSE IF( LDVT.LT.M ) THEN
*       INFO = -12
*   ELSE IF( LDU2.LT.N ) THEN
*       INFO = -15
*   ELSE IF( LDVT2.LT.M ) THEN
*       INFO = -17
*   END IF

```

```

      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DLASD2', -INFO )
        RETURN
      END IF
*
      NLP1 = NL + 1
      NLP2 = NL + 2
*
*      Generate the first part of the vector Z; and move the singular
*      values in the first part of D one position backward.
*
      Z1 = ALPHA*VT( NLP1, NLP1 )
      Z( 1 ) = Z1
      DO 10 I = NL, 1, -1
        Z( I+1 ) = ALPHA*VT( I, NLP1 )
        D( I+1 ) = D( I )
        IDXQ( I+1 ) = IDXQ( I ) + 1
10 CONTINUE
*
*      Generate the second part of the vector Z.
*
      DO 20 I = NLP2, M
        Z( I ) = BETA*VT( I, NLP2 )
20 CONTINUE
*
*      Initialize some reference arrays.
*
      DO 30 I = 2, NLP1
        COLTYP( I ) = 1
30 CONTINUE
      DO 40 I = NLP2, N
        COLTYP( I ) = 2
40 CONTINUE
*
*      Sort the singular values into increasing order
*
      DO 50 I = NLP2, N
        IDXQ( I ) = IDXQ( I ) + NLP1
50 CONTINUE
*
*      DSIGMA, IDXC, IDXC, and the first column of U2
*      are used as storage space.
*
      DO 60 I = 2, N
        DSIGMA( I ) = D( IDXQ( I ) )
        U2( I, 1 ) = Z( IDXQ( I ) )
        IDXC( I ) = COLTYP( IDXQ( I ) )
60 CONTINUE
*
      CALL DLAMRG( NL, NR, DSIGMA( 2 ), 1, 1, IDX( 2 ) )

```



```

*
      DO 70 I = 2, N
          IDXI = 1 + IDX( I )
          D( I ) = DSIGMA( IDXI )
          Z( I ) = U2( IDXI, 1 )
          COLTYP( I ) = IDXC( IDXI )
70 CONTINUE
*
*      Calculate the allowable deflation tolerance
*
      EPS = DLAMCH( 'Epsilon' )
      TOL = MAX( ABS( ALPHA ), ABS( BETA ) )
      TOL = EIGHT*EPS*MAX( ABS( D( N ) ), TOL )
*
*      There are 2 kinds of deflation -- first a value in the z-vector
*      is small, second two (or more) singular values are very close
*      together (their difference is small).
*
*      If the value in the z-vector is small, we simply permute the
*      array so that the corresponding singular value is moved to the
*      end.
*
*      If two values in the D-vector are close, we perform a two-sided
*      rotation designed to make one of the corresponding z-vector
*      entries zero, and then permute the array so that the deflated
*      singular value is moved to the end.
*
*      If there are multiple singular values then the problem deflates.
*      Here the number of equal singular values are found. As each equal
*      singular value is found, an elementary reflector is computed to
*      rotate the corresponding singular subspace so that the
*      corresponding components of Z are zero in this new basis.
*
      K = 1
      K2 = N + 1
      DO 80 J = 2, N
          IF( ABS( Z( J ) ).LE.TOL ) THEN
*
*              Deflate due to small z component.
*
              K2 = K2 - 1
              IDXP( K2 ) = J
              COLTYP( J ) = 4
              IF( J.EQ.N )
$                  GO TO 120
              ELSE
                  JPREV = J
                  GO TO 90
              END IF
80 CONTINUE

```

```

90 CONTINUE
  J = JPREV
100 CONTINUE
  J = J + 1
  IF( J.GT.N )
    $ GO TO 110
  IF( ABS( Z( J ) ).LE.TOL ) THEN
*
*     Deflate due to small z component.
*
    K2 = K2 - 1
    IDXP( K2 ) = J
    COLTYP( J ) = 4
  ELSE
*
*     Check if singular values are close enough to allow deflation.
*
    IF( ABS( D( J ) - D( JPREV ) ).LE.TOL ) THEN
*
*     Deflation is possible.
*
    S = Z( JPREV )
    C = Z( J )
*
*     Find sqrt(a**2+b**2) without overflow or
*     destructive underflow.
*
    TAU = DLAPY2( C, S )
    C = C / TAU
    S = -S / TAU
    Z( J ) = TAU
    Z( JPREV ) = ZERO
*
*     Apply back the Givens rotation to the left and right
*     singular vector matrices.
*
    IDXJP = IDXQ( IDX( JPREV ) + 1 )
    IDXJ = IDXQ( IDX( J ) + 1 )
    IF( IDXJP.LE.NLP1 ) THEN
      IDXJP = IDXJP - 1
    END IF
    IF( IDXJ.LE.NLP1 ) THEN
      IDXJ = IDXJ - 1
    END IF
    CALL DROT( N, U( 1, IDXJP ), 1, U( 1, IDXJ ), 1, C, S )
    CALL DROT( M, VT( IDXJP, 1 ), LDVT, VT( IDXJ, 1 ), LDVT, C,
$      S )
    IF( COLTYP( J ).NE.COLTYP( JPREV ) ) THEN
      COLTYP( J ) = 3
    END IF

```

```

        COLTYP( JPREV ) = 4
        K2 = K2 - 1
        IDXP( K2 ) = JPREV
        JPREV = J
    ELSE
        K = K + 1
        U2( K, 1 ) = Z( JPREV )
        DSIGMA( K ) = D( JPREV )
        IDXP( K ) = JPREV
        JPREV = J
    END IF
END IF
GO TO 100
110 CONTINUE
*
*   Record the last singular value.
*
        K = K + 1
        U2( K, 1 ) = Z( JPREV )
        DSIGMA( K ) = D( JPREV )
        IDXP( K ) = JPREV
*
120 CONTINUE
*
*   Count up the total number of the various types of columns, then
*   form a permutation which positions the four column types into
*   four groups of uniform structure (although one or more of these
*   groups may be empty).
*
        DO 130 J = 1, 4
            CTOT( J ) = 0
130 CONTINUE
        DO 140 J = 2, N
            CT = COLTYP( J )
            CTOT( CT ) = CTOT( CT ) + 1
140 CONTINUE
*
*   PSM(*) = Position in SubMatrix (of types 1 through 4)
*
        PSM( 1 ) = 2
        PSM( 2 ) = 2 + CTOT( 1 )
        PSM( 3 ) = PSM( 2 ) + CTOT( 2 )
        PSM( 4 ) = PSM( 3 ) + CTOT( 3 )
*
*   Fill out the IDXC array so that the permutation which it induces
*   will place all type-1 columns first, all type-2 columns next,
*   then all type-3's, and finally all type-4's, starting from the
*   second column. This applies similarly to the rows of VT.
*
        DO 150 J = 2, N

```

```

        JP = IDXP( J )
        CT = COLTYP( JP )
        IDXC( PSM( CT ) ) = J
        PSM( CT ) = PSM( CT ) + 1
150 CONTINUE
*
*      Sort the singular values and corresponding singular vectors into
*      DSIGMA, U2, and VT2 respectively. The singular values/vectors
*      which were not deflated go into the first K slots of DSIGMA, U2,
*      and VT2 respectively, while those which were deflated go into the
*      last N - K slots, except that the first column/row will be treated
*      separately.
*
      DO 160 J = 2, N
        JP = IDXP( J )
        DSIGMA( J ) = D( JP )
        IDXJ = IDXQ( IDX( IDXP( IDXC( J ) ) )+1 )
        IF( IDXJ.LE.NLP1 ) THEN
          IDXJ = IDXJ - 1
        END IF
        CALL DCOPY( N, U( 1, IDXJ ), 1, U2( 1, J ), 1 )
        CALL DCOPY( M, VT( IDXJ, 1 ), LDVT, VT2( J, 1 ), LDVT2 )
160 CONTINUE
*
*      Determine DSIGMA(1), DSIGMA(2) and Z(1)
*
      DSIGMA( 1 ) = ZERO
      HLFTOL = TOL / TWO
      IF( ABS( DSIGMA( 2 ) ).LE.HLFTOL )
$      DSIGMA( 2 ) = HLFTOL
      IF( M.GT.N ) THEN
        Z( 1 ) = DLAPY2( Z1, Z( M ) )
        IF( Z( 1 ).LE.TOL ) THEN
          C = ONE
          S = ZERO
          Z( 1 ) = TOL
        ELSE
          C = Z1 / Z( 1 )
          S = Z( M ) / Z( 1 )
        END IF
      ELSE
        IF( ABS( Z1 ).LE.TOL ) THEN
          Z( 1 ) = TOL
        ELSE
          Z( 1 ) = Z1
        END IF
      END IF
*
*      Move the rest of the updating row to Z.
*

```

```

      CALL DCOPY( K-1, U2( 2, 1 ), 1, Z( 2 ), 1 )
*
*   Determine the first column of U2, the first row of VT2 and the
*   last row of VT.
*
      CALL DLASET( 'A', N, 1, ZERO, ZERO, U2, LDU2 )
      U2( NLP1, 1 ) = ONE
      IF( M.GT.N ) THEN
        DO 170 I = 1, NLP1
          VT( M, I ) = -S*VT( NLP1, I )
          VT2( 1, I ) = C*VT( NLP1, I )
170    CONTINUE
        DO 180 I = NLP2, M
          VT2( 1, I ) = S*VT( M, I )
          VT( M, I ) = C*VT( M, I )
180    CONTINUE
      ELSE
        CALL DCOPY( M, VT( NLP1, 1 ), LDVT, VT2( 1, 1 ), LDVT2 )
      END IF
      IF( M.GT.N ) THEN
        CALL DCOPY( M, VT( M, 1 ), LDVT, VT2( M, 1 ), LDVT2 )
      END IF
*
*   The deflated singular values and their corresponding vectors go
*   into the back of D, U, and V respectively.
*
      IF( N.GT.K ) THEN
        CALL DCOPY( N-K, DSIGMA( K+1 ), 1, D( K+1 ), 1 )
        CALL DLACPY( 'A', N, N-K, U2( 1, K+1 ), LDU2, U( 1, K+1 ),
$          LDU )
        CALL DLACPY( 'A', N-K, M, VT2( K+1, 1 ), LDVT2, VT( K+1, 1 ),
$          LDVT )
      END IF
*
*   Copy CTOT into COLTYP for referencing in DLASD3.
*
      DO 190 J = 1, 4
        COLTYP( J ) = CTOT( J )
190    CONTINUE
*
      RETURN
*
*   End of DLASD2
*
      END

```

— LAPACK dlasd2 —

```

(let* ((zero 0.0) (one 1.0) (two 2.0) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 8.0 8.0) eight))
  (defun dlasd2
    (nl nr sqre k d z alpha beta u ldu vt ldvt dsigma u2 ldu2 vt2 ldvt2
     idxp idx idxc idxq coltyp info)
    (declare (type (simple-array fixnum (*)) coltyp idxq idxc idx idxp)
              (type (double-float) beta alpha)
              (type (simple-array double-float (*)) vt2 u2 dsigma vt u z d)
              (type fixnum info ldvt2 ldu2 ldvt ldu k sqre nr nl))
    (f2cl-lib:with-multi-array-data
      ((d double-float d-%data% d-%offset%)
       (z double-float z-%data% z-%offset%)
       (u double-float u-%data% u-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (dsigma double-float dsigma-%data% dsigma-%offset%)
       (u2 double-float u2-%data% u2-%offset%)
       (vt2 double-float vt2-%data% vt2-%offset%)
       (idxp fixnum idxp-%data% idxp-%offset%)
       (idx fixnum idx-%data% idx-%offset%)
       (idxc fixnum idxc-%data% idxc-%offset%)
       (idxq fixnum idxq-%data% idxq-%offset%)
       (coltyp fixnum coltyp-%data% coltyp-%offset%))
      (prog ((c 0.0) (eps 0.0) (hlftol 0.0) (s 0.0) (tau 0.0) (tol 0.0)
             (z1 0.0) (ct 0) (i 0) (idxi 0) (idxj 0) (idxjp 0) (j 0) (jp 0)
             (jprev 0) (k2 0) (m 0) (n 0) (nlp1 0) (nlp2 0)
             (ctot (make-array 4 :element-type 'fixnum))
             (psm (make-array 4 :element-type 'fixnum)))
             (declare (type (double-float) c eps hlftol s tau tol z1)
                       (type fixnum ct i idxi idxj idxjp j jp jprev k2 m
                                n nlp1 nlp2)
                       (type (simple-array fixnum (4)) ctot psm))
             (setf info 0)
             (cond
              ((< nl 1)
               (setf info -1))
              ((< nr 1)
               (setf info -2))
              ((and (/= sqre 1) (/= sqre 0))
               (setf info -3)))
             (setf n (f2cl-lib:int-add nl nr 1))
             (setf m (f2cl-lib:int-add n sqre))
             (cond
              ((< ldu n)
               (setf info -10))
              ((< ldvt m)

```

```

      (setf info -12))
    (< ldu2 n)
    (setf info -15))
    (< ldvt2 m)
    (setf info -17)))
  (cond
    ((/= info 0)
     (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DLASD2" (f2cl-lib:int-sub info))
     (go end_label)))
  (setf nlp1 (f2cl-lib:int-add nl 1))
  (setf nlp2 (f2cl-lib:int-add nl 2))
  (setf z1
    (* alpha
      (f2cl-lib:fref vt-%data%
                     (nlp1 nlp1)
                     ((1 ldvt) (1 *))
                     vt-%offset%)))
    (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1)
    (f2cl-lib:fdo (i nl (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
      (> i 1) nil)
    (tagbody
      (setf (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add i 1))
                          ((1 *))
                          z-%offset%)
        (* alpha
          (f2cl-lib:fref vt-%data%
                          (i nlp1)
                          ((1 ldvt) (1 *))
                          vt-%offset%)))
        (setf (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-add i 1))
                            ((1 *))
                            d-%offset%)
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
        (setf (f2cl-lib:fref idxq-%data%
                            ((f2cl-lib:int-add i 1))
                            ((1 *))
                            idxq-%offset%)
          (f2cl-lib:int-add
            (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
            1))))
    (f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
      (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (* beta
          (f2cl-lib:fref vt-%data%

```

```

                                (i nlp2)
                                ((1 ldvt) (1 *))
                                vt-%offset%))))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              (> i nlp1) nil)
  (tagbody
    (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%) 1)))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%) 2)))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
              (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
          (f2cl-lib:int-add
            (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
            nlp1))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              (> i n) nil)
  (tagbody
    (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
          (f2cl-lib:fref d-%data%
                        ((f2cl-lib:fref idxq (i) ((1 *)))
                         ((1 *))
                         d-%offset%)))
    (setf (f2cl-lib:fref u2-%data% (i 1) ((1 ldu2) (1 *)) u2-%offset%)
          (f2cl-lib:fref z-%data%
                        ((f2cl-lib:fref idxq (i) ((1 *)))
                         ((1 *))
                         z-%offset%)))
    (setf (f2cl-lib:fref idxc-%data% (i) ((1 *)) idxc-%offset%)
          (f2cl-lib:fref coltyp-%data%
                        ((f2cl-lib:fref idxq (i) ((1 *)))
                         ((1 *))
                         coltyp-%offset%))))))
(dlamrg nl nr (f2cl-lib:array-slice dsigma double-float (2) ((1 *))) 1
  1 (f2cl-lib:array-slice idx fixnum (2) ((1 *))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              (> i n) nil)
  (tagbody
    (setf idxi
          (f2cl-lib:int-add 1
                            (f2cl-lib:fref idx-%data%
                                              (i)
                                              ((1 *))
                                              idx-%offset%)))
    (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          (f2cl-lib:fref dsigma-%data%
                        (idxi)

```



```

                                ((1 *))
                                dsigma-%offset%))
  (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (f2cl-lib:fref u2-%data%
                        (idxi 1)
                        ((1 ldu2) (1 *))
                        u2-%offset%))
  (setf (f2cl-lib:fref coltyp-%data% (i) ((1 *)) coltyp-%offset%)
        (f2cl-lib:fref idxc-%data% (idxi) ((1 *)) idxc-%offset%)))
  (setf eps (dlamch "Epsilon"))
  (setf tol (max (abs alpha) (abs beta)))
  (setf tol
    (* eight
      eps
      (max (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
            tol)))
  (setf k 1)
  (setf k2 (f2cl-lib:int-add n 1))
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
        (setf k2 (f2cl-lib:int-sub k2 1))
        (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
        (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%)
              4)
        (if (= j n) (go label120)))
      (t
        (setf jprev j)
        (go label90)))))
label90
  (setf j jprev)
label100
  (setf j (f2cl-lib:int-add j 1))
  (if (> j n) (go label110))
  (cond
    ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
      (setf k2 (f2cl-lib:int-sub k2 1))
      (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
      (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%) 4))
    (t
      (cond
        ((<=
          (abs
            (+ (f2cl-lib:fref d (j) ((1 *)))
              (- (f2cl-lib:fref d (jprev) ((1 *))))))
          tol)
          (setf s (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
          (setf c (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))

```

```

(setf tau (dlapy2 c s))
(setf c (/ c tau))
(setf s (/ (- s) tau))
(setf (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%) tau)
(setf (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%) zero)
(setf idxjp
  (f2cl-lib:fref idxq-%data%
    ((f2cl-lib:int-add
      (f2cl-lib:fref idx (jprev) ((1 *))
      1))
    ((1 *))
    idxq-%offset%))
(setf idxj
  (f2cl-lib:fref idxq-%data%
    ((f2cl-lib:int-add
      (f2cl-lib:fref idx (j) ((1 *))
      1))
    ((1 *))
    idxq-%offset%))
(cond
  ((<= idxjp nlp1)
    (setf idxjp (f2cl-lib:int-sub idxjp 1))))
(cond
  ((<= idxj nlp1)
    (setf idxj (f2cl-lib:int-sub idxj 1))))
(drot n
  (f2cl-lib:array-slice u double-float (1 idxjp) ((1 ldu) (1 *)))
  1 (f2cl-lib:array-slice u double-float (1 idxj) ((1 ldu) (1 *)))
  1 c s)
(drot m
  (f2cl-lib:array-slice vt
    double-float
    (idxjp 1)
    ((1 ldvt) (1 *)))

  ldvt
  (f2cl-lib:array-slice vt double-float (idxj 1) ((1 ldvt) (1 *)))
  ldvt c s)
(cond
  ((/= (f2cl-lib:fref coltyp (j) ((1 *)))
    (f2cl-lib:fref coltyp (jprev) ((1 *))))
    (setf (f2cl-lib:fref coltyp-%data%
      (j)
      ((1 *))
      coltyp-%offset%)
      3)))
(setf (f2cl-lib:fref coltyp-%data%
  (jprev)
  ((1 *))
  coltyp-%offset%)
  4)

```

```

      (setf k2 (f2cl-lib:int-sub k2 1))
      (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%)
            jprev)
      (setf jprev j))
    (t
      (setf k (f2cl-lib:int-add k 1))
      (setf (f2cl-lib:fref u2-%data%
                          (k 1)
                          ((1 ldu2) (1 *))
                          u2-%offset%)
            (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
      (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
            (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
      (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
      (setf jprev j))))
    (go label100)
label110
  (setf k (f2cl-lib:int-add k 1))
  (setf (f2cl-lib:fref u2-%data% (k 1) ((1 ldu2) (1 *)) u2-%offset%)
        (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
  (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
        (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
label120
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                (> j 4) nil)
  (tagbody (setf (f2cl-lib:fref ctot (j) ((1 4))) 0)))
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                (> j n) nil)
  (tagbody
    (setf ct (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%))
    (setf (f2cl-lib:fref ctot (ct) ((1 4)))
          (f2cl-lib:int-add (f2cl-lib:fref ctot (ct) ((1 4))) 1))))
  (setf (f2cl-lib:fref psm (1) ((1 4))) 2)
  (setf (f2cl-lib:fref psm (2) ((1 4)))
        (f2cl-lib:int-add 2 (f2cl-lib:fref ctot (1) ((1 4)))))
  (setf (f2cl-lib:fref psm (3) ((1 4)))
        (f2cl-lib:int-add (f2cl-lib:fref psm (2) ((1 4)))
                          (f2cl-lib:fref ctot (2) ((1 4)))))
  (setf (f2cl-lib:fref psm (4) ((1 4)))
        (f2cl-lib:int-add (f2cl-lib:fref psm (3) ((1 4)))
                          (f2cl-lib:fref ctot (3) ((1 4)))))
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
                (> j n) nil)
  (tagbody
    (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf ct
          (f2cl-lib:fref coltyp-%data% (jp) ((1 *)) coltyp-%offset%))
    (setf (f2cl-lib:fref idxc-%data%
                      ((f2cl-lib:fref psm (ct) ((1 4)))))
          (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%)
          (f2cl-lib:fref idxc-%data%
                      ((f2cl-lib:fref psm (ct) ((1 4)))))
          (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf jprev j)
    (go label110)
  )

```

```

                                ((1 *))
                                idxc-%offset%)
                                j)
    (setf (f2cl-lib:fref psm (ct) ((1 4)))
          (f2cl-lib:int-add (f2cl-lib:fref psm (ct) ((1 4))) 1)))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
              ((> j n) nil)
  (tagbody
    (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
          (f2cl-lib:fref d-%data% (jp) ((1 *)) d-%offset%))
    (setf idxj
          (f2cl-lib:fref idxq-%data%
            ((f2cl-lib:int-add
              (f2cl-lib:fref idx
                ((f2cl-lib:fref idxp
                  ((f2cl-lib:fref
                    idxc
                    (j)
                    ((1 *))))
                  ((1 *))))
                ((1 *)))
              1))
            ((1 *))
            idxq-%offset%))
    (cond
      ((<= idxj nlp1)
       (setf idxj (f2cl-lib:int-sub idxj 1))))
    (dcopy n
      (f2cl-lib:array-slice u double-float (1 idxj) ((1 ldu) (1 *))) 1
      (f2cl-lib:array-slice u2 double-float (1 j) ((1 ldu2) (1 *))) 1)
    (dcopy m
      (f2cl-lib:array-slice vt double-float (idxj 1) ((1 ldvt) (1 *)))
      ldvt
      (f2cl-lib:array-slice vt2 double-float (j 1) ((1 ldvt2) (1 *)))
      ldvt2)))
    (setf (f2cl-lib:fref dsigma-%data% (1) ((1 *)) dsigma-%offset%) zero)
    (setf hlftol (/ tol two))
    (if
      (<= (abs (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%))
          hlftol)
      (setf (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%)
            hlftol))
    (cond
      ((> m n)
       (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
             (dlapy2 z1 (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)))
       (cond
         ((<= (f2cl-lib:fref z (1) ((1 *))) tol)
          (setf c one)

```

```

      (setf s zero)
      (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
    (t
      (setf c (/ z1 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
      (setf s
        (/ (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)
           (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))))))
  (t
    (cond
      ((<= (abs z1) tol)
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
      (t
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1))))))
  (dcopy (f2cl-lib:int-sub k 1)
    (f2cl-lib:array-slice u2 double-float (2 1) ((1 ldu2) (1 *))) 1
    (f2cl-lib:array-slice z double-float (2) ((1 *))) 1)
  (dlaset "A" n 1 zero zero u2 ldu2)
  (setf (f2cl-lib:fref u2-%data% (nlp1 1) ((1 ldu2) (1 *)) u2-%offset%)
    one)
  (cond
    ((> m n)
      (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
        ((> i nlp1) nil)
        (tagbody
          (setf (f2cl-lib:fref vt-%data%
            (m i)
            ((1 ldvt) (1 *))
            vt-%offset%)
            (* (- s)
              (f2cl-lib:fref vt-%data%
                (nlp1 i)
                ((1 ldvt) (1 *))
                vt-%offset%)))
          (setf (f2cl-lib:fref vt2-%data%
            (1 i)
            ((1 ldvt2) (1 *))
            vt2-%offset%)
            (* c
              (f2cl-lib:fref vt-%data%
                (nlp1 i)
                ((1 ldvt) (1 *))
                vt-%offset%))))))
      (f2cl-lib:fd0 (i nlp2 (f2cl-lib:int-add i 1))
        ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref vt2-%data%
            (1 i)
            ((1 ldvt2) (1 *))
            vt2-%offset%)
            (* s

```

```

(f2cl-lib:fref vt-%data%
  (m i)
  ((1 ldvt) (1 *)))
  vt-%offset%)))
(setf (f2cl-lib:fref vt-%data%
  (m i)
  ((1 ldvt) (1 *)))
  vt-%offset%)
(* c
  (f2cl-lib:fref vt-%data%
    (m i)
    ((1 ldvt) (1 *)))
    vt-%offset%))))))
(t
  (dcopy m
    (f2cl-lib:array-slice vt double-float (nlp1 1) ((1 ldvt) (1 *)))
    ldvt
    (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *)))
    ldvt2)))
(cond
  ((> m n)
    (dcopy m
      (f2cl-lib:array-slice vt double-float (m 1) ((1 ldvt) (1 *))) ldvt
      (f2cl-lib:array-slice vt2 double-float (m 1) ((1 ldvt2) (1 *)))
      ldvt2)))
  (cond
    ((> n k)
      (dcopy (f2cl-lib:int-sub n k)
        (f2cl-lib:array-slice dsigma double-float ((+ k 1) ((1 *))) 1
        (f2cl-lib:array-slice d double-float ((+ k 1) ((1 *))) 1)
        (dlacpy "A" n (f2cl-lib:int-sub n k)
          (f2cl-lib:array-slice u2
            double-float
            (1 (f2cl-lib:int-add k 1))
            ((1 ldu2) (1 *)))
          ldu2
          (f2cl-lib:array-slice u
            double-float
            (1 (f2cl-lib:int-add k 1))
            ((1 ldu) (1 *)))
          ldu)
        (dlacpy "A" (f2cl-lib:int-sub n k) m
          (f2cl-lib:array-slice vt2
            double-float
            ((+ k 1) 1)
            ((1 ldvt2) (1 *)))
          ldvt2
          (f2cl-lib:array-slice vt double-float ((+ k 1) 1) ((1 ldvt) (1 *)))
          ldvt)))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

                                (> j 4) nil)
      (tagbody
        (setf (f2cl-lib:fref coltyp-%data% (j) ((1 *)) coltyp-%offset%)
              (f2cl-lib:fref ctot (j) ((1 4)))))
end_label
(return
 (values nil
         nil
         nil
         k
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         nil
         info))))))

```

dlasd3 LAPACK

— dlasd3.input —

```

)set break resume
)sys rm -f dlasd3.output
)spool dlasd3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd3.help —

```
=====
dlasd3 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASD3 - all the square roots of the roots of the secular equation, as defined by the values in D and Z

SYNOPSIS

```
SUBROUTINE DLASD3( NL, NR, SQRE, K, D, Q, LDQ, DSIGMA, U, LDU, U2,
                  LDU2, VT, LDVT, VT2, LDVT2, IDXC, CTOT, Z, INFO )
```

```
      INTEGER      INFO, K, LDQ, LDU, LDU2, LDVT, LDVT2, NL, NR, SQRE
```

```
      INTEGER      CTOT( * ), IDXC( * )
```

```
      DOUBLE      PRECISION D( * ), DSIGMA( * ), Q( LDQ, * ), U( LDU,
* ), U2( LDU2, * ), VT( LDVT, * ), VT2( LDVT2, * ),
Z( * )
```

Purpose

```
=====
```

DLASD3 finds all the square roots of the roots of the secular equation, as defined by the values in D and Z. It makes the appropriate calls to DLASD4 and then updates the singular vectors by matrix multiplication.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray XMP, Cray YMP, Cray C 90, or Cray 2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

DLASD3 is called from DLASD1.

Arguments

```
=====
```

NL (input) INTEGER

The row dimension of the upper block. $NL \geq 1$.

NR (input) INTEGER
The row dimension of the lower block. $NR \geq 1$.

SQRE (input) INTEGER
= 0: the lower block is an NR -by- NR square matrix.
= 1: the lower block is an NR -by- $(NR+1)$ rectangular matrix.

The bidiagonal matrix has $N = NL + NR + 1$ rows and
 $M = N + SQRE \geq N$ columns.

K (input) INTEGER
The size of the secular equation, $1 \leq K \leq N$.

D (output) DOUBLE PRECISION array, dimension(K)
On exit the square roots of the roots of the secular equation,
in ascending order.

Q (workspace) DOUBLE PRECISION array,
dimension at least (LDQ, K) .

LDQ (input) INTEGER
The leading dimension of the array Q. $LDQ \geq K$.

DSIGMA (input) DOUBLE PRECISION array, dimension(K)
The first K elements of this array contain the old roots
of the deflated updating problem. These are the poles
of the secular equation.

U (input) DOUBLE PRECISION array, dimension (LDU, N)
The last $N - K$ columns of this matrix contain the deflated
left singular vectors.

LDU (input) INTEGER
The leading dimension of the array U. $LDU \geq N$.

U2 (input) DOUBLE PRECISION array, dimension $(LDU2, N)$
The first K columns of this matrix contain the non-deflated
left singular vectors for the split problem.

LDU2 (input) INTEGER
The leading dimension of the array U2. $LDU2 \geq N$.

VT (input) DOUBLE PRECISION array, dimension $(LDVT, M)$
The last $M - K$ columns of VT' contain the deflated
right singular vectors.

LDVT (input) INTEGER
The leading dimension of the array VT. $LDVT \geq N$.

VT2 (input) DOUBLE PRECISION array, dimension (LDVT2, N)
 The first K columns of VT2' contain the non-deflated
 right singular vectors for the split problem.

LDVT2 (input) INTEGER
 The leading dimension of the array VT2. LDVT2 \geq N.

IDXC (input) INTEGER array, dimension (N)
 The permutation used to arrange the columns of U (and rows of
 VT) into three groups: the first group contains non-zero
 entries only at and above (or before) NL +1; the second
 contains non-zero entries only at and below (or after) NL+2;
 and the third is dense. The first column of U and the row of
 VT are treated separately, however.

The rows of the singular vectors found by DLASD4
 must be likewise permuted before the matrix multiplies can
 take place.

CTOT (input) INTEGER array, dimension (4)
 A count of the total number of the various types of columns
 in U (or rows in VT), as described in IDXC. The fourth column
 type is any column which has been deflated.

Z (input) DOUBLE PRECISION array, dimension (K)
 The first K elements of this array contain the components
 of the deflation-adjusted updating row vector.

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: if INFO = 1, an singular value did not converge

Further Details
 =====

Based on contributions by
 Ming Gu and Huan Ren, Computer Science Division, University of
 California at Berkeley, USA

— dlasd3.f —

```

SUBROUTINE DLASD3( NL, NR, SQRE, K, D, Q, LDQ, DSIGMA, U, LDU, U2,
$                  LDU2, VT, LDVT, VT2, LDVT2, IDXC, CTOT, Z,
$                  INFO )

```

```

*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*    Courant Institute, NAG Ltd., and Rice University
*    October 31, 1999
*
*    .. Scalar Arguments ..
      INTEGER          INFO, K, LDQ, LDU, LDU2, LDVT, LDVT2, NL, NR,
$                   SQRE
*
*    ..
*
*    .. Array Arguments ..
      INTEGER          CTOT( * ), IDXC( * )
      DOUBLE PRECISION D( * ), DSIGMA( * ), Q( LDQ, * ), U( LDU, * ),
$                   U2( LDU2, * ), VT( LDVT, * ), VT2( LDVT2, * ),
$                   Z( * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
      DOUBLE PRECISION ONE, ZERO, NEGONE
      PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0,
$                   NEGONE = -1.0D+0 )
*
*    ..
*
*    .. Local Scalars ..
      INTEGER          CTEMP, I, J, JC, KTEMP, M, N, NLP1, NLP2, NRP1
      DOUBLE PRECISION RHO, TEMP
*
*    ..
*
*    .. External Functions ..
      DOUBLE PRECISION DLAMC3, DNRM2
      EXTERNAL         DLAMC3, DNRM2
*
*    ..
*
*    .. External Subroutines ..
      EXTERNAL         DCOPY, DGEMM, DLACPY, DLASCL, DLASD4, XERBLA
*
*    ..
*
*    .. Intrinsic Functions ..
      INTRINSIC        ABS, SIGN, SQRT
*
*    ..
*
*    .. Executable Statements ..
*
*    Test the input parameters.
*
*
      INFO = 0
*
      IF( NL.LT.1 ) THEN
        INFO = -1
      ELSE IF( NR.LT.1 ) THEN
        INFO = -2
      ELSE IF( ( SQRE.NE.1 ) .AND. ( SQRE.NE.0 ) ) THEN
        INFO = -3

```

```

      END IF
*
      N = NL + NR + 1
      M = N + SQRE
      NLP1 = NL + 1
      NLP2 = NL + 2
*
      IF( ( K.LT.1 ) .OR. ( K.GT.N ) ) THEN
          INFO = -4
      ELSE IF( LDQ.LT.K ) THEN
          INFO = -7
      ELSE IF( LDU.LT.N ) THEN
          INFO = -10
      ELSE IF( LDU2.LT.N ) THEN
          INFO = -12
      ELSE IF( LDVT.LT.M ) THEN
          INFO = -14
      ELSE IF( LDVT2.LT.M ) THEN
          INFO = -16
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'DLASD3', -INFO )
          RETURN
      END IF
*
*   Quick return if possible
*
      IF( K.EQ.1 ) THEN
          D( 1 ) = ABS( Z( 1 ) )
          CALL DCOPY( M, VT2( 1, 1 ), LDVT2, VT( 1, 1 ), LDVT )
          IF( Z( 1 ).GT.ZERO ) THEN
              CALL DCOPY( N, U2( 1, 1 ), 1, U( 1, 1 ), 1 )
          ELSE
              DO 10 I = 1, N
                  U( I, 1 ) = -U2( I, 1 )
10          CONTINUE
          END IF
          RETURN
      END IF
*
*   Modify values DSIGMA(i) to make sure all DSIGMA(i)-DSIGMA(j) can
*   be computed with high relative accuracy (barring over/underflow).
*   This is a problem on machines without a guard digit in
*   add/subtract (Cray XMP, Cray YMP, Cray C 90 and Cray 2).
*   The following code replaces DSIGMA(I) by 2*DSIGMA(I)-DSIGMA(I),
*   which on any of these machines zeros out the bottommost
*   bit of DSIGMA(I) if it is 1; this makes the subsequent
*   subtractions DSIGMA(I)-DSIGMA(J) unproblematic when cancellation
*   occurs. On binary machines with a guard digit (almost all
*   machines) it does not change DSIGMA(I) at all. On hexadecimal

```

```

*      and decimal machines with a guard digit, it slightly
*      changes the bottommost bits of DSIGMA(I). It does not account
*      for hexadecimal or decimal machines without guard digits
*      (we know of none). We use a subroutine call to compute
*      2*DLAMBDA(I) to prevent optimizing compilers from eliminating
*      this code.
*
      DO 20 I = 1, K
          DSIGMA( I ) = DLAMC3( DSIGMA( I ), DSIGMA( I ) ) - DSIGMA( I )
20  CONTINUE
*
*      Keep a copy of Z.
*
      CALL DCOPY( K, Z, 1, Q, 1 )
*
*      Normalize Z.
*
      RHO = DNRM2( K, Z, 1 )
      CALL DLASCL( 'G', 0, 0, RHO, ONE, K, 1, Z, K, INFO )
      RHO = RHO*RHO
*
*      Find the new singular values.
*
      DO 30 J = 1, K
          CALL DLASD4( K, J, DSIGMA, Z, U( 1, J ), RHO, D( J ),
$              VT( 1, J ), INFO )
*
*      If the zero finder fails, the computation is terminated.
*
      IF( INFO.NE.0 ) THEN
          RETURN
      END IF
30  CONTINUE
*
*      Compute updated Z.
*
      DO 60 I = 1, K
          Z( I ) = U( I, K )*VT( I, K )
          DO 40 J = 1, I - 1
              Z( I ) = Z( I )*( U( I, J )*VT( I, J ) /
$                  ( DSIGMA( I )-DSIGMA( J ) ) ) /
$                  ( DSIGMA( I )+DSIGMA( J ) ) )
40      CONTINUE
          DO 50 J = I, K - 1
              Z( I ) = Z( I )*( U( I, J )*VT( I, J ) /
$                  ( DSIGMA( I )-DSIGMA( J+1 ) ) ) /
$                  ( DSIGMA( I )+DSIGMA( J+1 ) ) )
50      CONTINUE
          Z( I ) = SIGN( SQRT( ABS( Z( I ) ) ), Q( I, 1 ) )
60  CONTINUE

```

```

*
*   Compute left singular vectors of the modified diagonal matrix,
*   and store related information for the right singular vectors.
*
      DO 90 I = 1, K
        VT( 1, I ) = Z( 1 ) / U( 1, I ) / VT( 1, I )
        U( 1, I ) = NEGONE
        DO 70 J = 2, K
          VT( J, I ) = Z( J ) / U( J, I ) / VT( J, I )
          U( J, I ) = DSIGMA( J ) * VT( J, I )
70      CONTINUE
        TEMP = DNRM2( K, U( 1, I ), 1 )
        Q( 1, I ) = U( 1, I ) / TEMP
        DO 80 J = 2, K
          JC = IDXC( J )
          Q( J, I ) = U( JC, I ) / TEMP
80      CONTINUE
90      CONTINUE

*
*   Update the left singular vector matrix.
*
      IF( K.EQ.2 ) THEN
        CALL DGEMM( 'N', 'N', N, K, K, ONE, U2, LDU2, Q, LDQ, ZERO, U,
$          LDU )
        GO TO 100
      END IF
      IF( CTOT( 1 ).GT.0 ) THEN
        CALL DGEMM( 'N', 'N', NL, K, CTOT( 1 ), ONE, U2( 1, 2 ), LDU2,
$          Q( 2, 1 ), LDQ, ZERO, U( 1, 1 ), LDU )
        IF( CTOT( 3 ).GT.0 ) THEN
          KTEMP = 2 + CTOT( 1 ) + CTOT( 2 )
          CALL DGEMM( 'N', 'N', NL, K, CTOT( 3 ), ONE, U2( 1, KTEMP ),
$          LDU2, Q( KTEMP, 1 ), LDQ, ONE, U( 1, 1 ), LDU )
        END IF
      ELSE IF( CTOT( 3 ).GT.0 ) THEN
        KTEMP = 2 + CTOT( 1 ) + CTOT( 2 )
        CALL DGEMM( 'N', 'N', NL, K, CTOT( 3 ), ONE, U2( 1, KTEMP ),
$          LDU2, Q( KTEMP, 1 ), LDQ, ZERO, U( 1, 1 ), LDU )
      ELSE
        CALL DLACPY( 'F', NL, K, U2, LDU2, U, LDU )
      END IF
      CALL DCOPY( K, Q( 1, 1 ), LDQ, U( NLP1, 1 ), LDU )
      KTEMP = 2 + CTOT( 1 )
      CTEMP = CTOT( 2 ) + CTOT( 3 )
      CALL DGEMM( 'N', 'N', NR, K, CTEMP, ONE, U2( NLP2, KTEMP ), LDU2,
$          Q( KTEMP, 1 ), LDQ, ZERO, U( NLP2, 1 ), LDU )

*
*   Generate the right singular vectors.
*
100      CONTINUE

```

```

      DO 120 I = 1, K
        TEMP = DNRM2( K, VT( 1, I ), 1 )
        Q( I, 1 ) = VT( 1, I ) / TEMP
        DO 110 J = 2, K
          JC = IDXC( J )
          Q( I, J ) = VT( JC, I ) / TEMP
110      CONTINUE
120      CONTINUE
*
*      Update the right singular vector matrix.
*
      IF( K.EQ.2 ) THEN
        CALL DGEMM( 'N', 'N', K, M, K, ONE, Q, LDQ, VT2, LDVT2, ZERO,
$              VT, LDVT )
        RETURN
      END IF
      KTEMP = 1 + CTOT( 1 )
      CALL DGEMM( 'N', 'N', K, NLP1, KTEMP, ONE, Q( 1, 1 ), LDQ,
$              VT2( 1, 1 ), LDVT2, ZERO, VT( 1, 1 ), LDVT )
      KTEMP = 2 + CTOT( 1 ) + CTOT( 2 )
      IF( KTEMP.LE.LDVT2 )
$        CALL DGEMM( 'N', 'N', K, NLP1, CTOT( 3 ), ONE, Q( 1, KTEMP ),
$              LDQ, VT2( KTEMP, 1 ), LDVT2, ONE, VT( 1, 1 ),
$              LDVT )
*
      KTEMP = CTOT( 1 ) + 1
      NRP1 = NR + SQRE
      IF( KTEMP.GT.1 ) THEN
        DO 130 I = 1, K
          Q( I, KTEMP ) = Q( I, 1 )
130      CONTINUE
        DO 140 I = NLP2, M
          VT2( KTEMP, I ) = VT2( 1, I )
140      CONTINUE
      END IF
      CTEMP = 1 + CTOT( 2 ) + CTOT( 3 )
      CALL DGEMM( 'N', 'N', K, NRP1, CTEMP, ONE, Q( 1, KTEMP ), LDQ,
$              VT2( KTEMP, NLP2 ), LDVT2, ZERO, VT( 1, NLP2 ), LDVT )
*
      RETURN
*
*      End of DLASD3
*
      END

```

```

(let* ((one 1.0) (zero 0.0) (negone (- 1.0)))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero)
            (type (double-float) negone))
  (defun dlasd3
    (nl nr sqre k d q ldq dsigma u ldu u2 ldu2 vt ldvt vt2 ldvt2 idxc ctot
     z info)
    (declare (type (simple-array fixnum (*)) ctot idxc)
              (type (simple-array double-float (*)) z vt2 vt u2 u dsigma q d)
              (type fixnum info ldvt2 ldvt ldu2 ldu ldq k sqre nr
                    nl))
    (f2cl-lib:with-multi-array-data
      ((d double-float d-%data% d-%offset%)
       (q double-float q-%data% q-%offset%)
       (dsigma double-float dsigma-%data% dsigma-%offset%)
       (u double-float u-%data% u-%offset%)
       (u2 double-float u2-%data% u2-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (vt2 double-float vt2-%data% vt2-%offset%)
       (z double-float z-%data% z-%offset%)
       (idxc fixnum idxc-%data% idxc-%offset%)
       (ctot fixnum ctot-%data% ctot-%offset%))
      (prog ((rho 0.0) (temp 0.0) (ctemp 0) (i 0) (j 0) (jc 0) (ktemp 0) (m 0)
             (n 0) (nlp1 0) (nlp2 0) (nrp1 0))
        (declare (type (double-float) rho temp)
                  (type fixnum ctemp i j jc ktemp m n nlp1 nlp2
                        nrp1))

        (setf info 0)
        (cond
          ((< nl 1)
            (setf info -1))
          ((< nr 1)
            (setf info -2))
          ((and (/= sqre 1) (/= sqre 0))
            (setf info -3)))
        (setf n (f2cl-lib:int-add nl nr 1))
        (setf m (f2cl-lib:int-add n sqre))
        (setf nlp1 (f2cl-lib:int-add nl 1))
        (setf nlp2 (f2cl-lib:int-add nl 2))
        (cond
          ((or (< k 1) (> k n))
            (setf info -4))
          ((< ldq k)
            (setf info -7))
          ((< ldu n)
            (setf info -10))
          ((< ldu2 n)
            (setf info -12))
          ((< ldvt m)
            (setf info -14))

```



```

      ((< ldvt2 m)
       (setf info -16)))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD3" (f2cl-lib:int-sub info))
   (go end_label)))
(cond
  ((= k 1)
   (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
         (abs (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
   (dcopy m
    (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *)))
    ldvt2 (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
    ldvt)
   (cond
    ((> (f2cl-lib:fref z (1) ((1 *))) zero)
     (dcopy n
      (f2cl-lib:array-slice u2 double-float (1 1) ((1 ldu2) (1 *))) 1
      (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *))) 1))
    (t
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      ((> i n) nil)
      (tagbody
       (setf (f2cl-lib:fref u-%data%
                           (i 1)
                           ((1 ldu) (1 *))
                           u-%offset%)
              (-
               (f2cl-lib:fref u2-%data%
                               (i 1)
                               ((1 ldu2) (1 *))
                               u2-%offset%))))))
     (go end_label)))
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i k) nil)
    (tagbody
     (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
           (-
            (multiple-value-bind (ret-val var-0 var-1)
              (dlamc3
               (f2cl-lib:fref dsigma-%data%
                               (i)
                               ((1 *))
                               dsigma-%offset%)
               (f2cl-lib:fref dsigma-%data%
                               (i)
                               ((1 *))
                               dsigma-%offset%))
            ret-val))))))

```

```

        (declare (ignore))
        (setf (f2cl-lib:fref dsigma-%data%
                           (i)
                           ((1 *))
                           dsigma-%offset%))
        var-0)
        (setf (f2cl-lib:fref dsigma-%data%
                           (i)
                           ((1 *))
                           dsigma-%offset%))
        var-1)
        ret-val)
        (f2cl-lib:fref dsigma-%data%
                       (i)
                       ((1 *))
                       dsigma-%offset%))))))
(dcopy k z 1 q 1)
(setf rho (dnrm2 k z 1))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 rho one k 1 z k info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                    var-8))
  (setf info var-9))
(setf rho (* rho rho))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dlasd4 k j dsigma z
            (f2cl-lib:array-slice u double-float (1 j) ((1 ldu) (1 *)))
            rho (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
            (f2cl-lib:array-slice vt double-float (1 j) ((1 ldvt) (1 *)))
            info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
    (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) var-6)
    (setf info var-8))
  (cond
    ((/= info 0)
     (go end_label))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i k) nil)
(tagbody
  (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (*
         (f2cl-lib:fref u-%data% (i k) ((1 ldu) (1 *)) u-%offset%)
         (f2cl-lib:fref vt-%data%
                        (i k)
                        ((1 ldvt) (1 *)))

```

```

                                vt-%offset%)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  ((> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (/
          (/
            (*
              (f2cl-lib:fref u-%data%
                (i j)
                ((1 ldu) (1 *))
                u-%offset%)
              (f2cl-lib:fref vt-%data%
                (i j)
                ((1 ldvt) (1 *))
                vt-%offset%)))
            (-
              (f2cl-lib:fref dsigma-%data%
                (i)
                ((1 *))
                dsigma-%offset%)
              (f2cl-lib:fref dsigma-%data%
                (j)
                ((1 *))
                dsigma-%offset%))))
          (+
            (f2cl-lib:fref dsigma-%data%
              (i)
              ((1 *))
              dsigma-%offset%)
            (f2cl-lib:fref dsigma-%data%
              (j)
              ((1 *))
              dsigma-%offset%))))))
(f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
  ((> j (f2cl-lib:int-add k (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (/
          (/
            (*
              (f2cl-lib:fref u-%data%
                (i j)
                ((1 ldu) (1 *))
                u-%offset%)
              (f2cl-lib:fref vt-%data%
                (i j)
                ((1 ldvt) (1 *))

```

```

                                vt-%offset%))
(-
  (f2cl-lib:fref dsigma-%data%
    (i)
    ((1 *))
    dsigma-%offset%)
  (f2cl-lib:fref dsigma-%data%
    ((f2cl-lib:int-add j 1))
    ((1 *))
    dsigma-%offset%)))
(+
  (f2cl-lib:fref dsigma-%data%
    (i)
    ((1 *))
    dsigma-%offset%)
  (f2cl-lib:fref dsigma-%data%
    ((f2cl-lib:int-add j 1))
    ((1 *))
    dsigma-%offset%))))))
(setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
  (f2cl-lib:sign
    (f2cl-lib:fsqrt
      (abs (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)))
    (f2cl-lib:fref q-%data%
      (i 1)
      ((1 ldq) (1 *))
      q-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i k) nil)
(tagbody
  (setf (f2cl-lib:fref vt-%data% (1 i) ((1 ldvt) (1 *)) vt-%offset%)
    (/
      (/ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
        (f2cl-lib:fref u-%data%
          (1 i)
          ((1 ldu) (1 *))
          u-%offset%))
      (f2cl-lib:fref vt-%data%
        (1 i)
        ((1 ldvt) (1 *))
        vt-%offset%)))
    (setf (f2cl-lib:fref u-%data% (1 i) ((1 ldu) (1 *)) u-%offset%)
      negone)
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
    (> j k) nil)
  (tagbody
    (setf (f2cl-lib:fref vt-%data%
      (j i)
      ((1 ldvt) (1 *))
      vt-%offset%)

```

```

(/
  (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
     (f2cl-lib:fref u-%data%
                    (j i)
                    ((1 ldu) (1 *))
                    u-%offset%))
    (f2cl-lib:fref vt-%data%
                  (j i)
                  ((1 ldvt) (1 *))
                  vt-%offset%)))
(setf (f2cl-lib:fref u-%data% (j i) ((1 ldu) (1 *)) u-%offset%)
      (*
        (f2cl-lib:fref dsigma-%data%
                        (j)
                        ((1 *))
                        dsigma-%offset%)
        (f2cl-lib:fref vt-%data%
                        (j i)
                        ((1 ldvt) (1 *))
                        vt-%offset%))))))
(setf temp
      (dnrm2 k
        (f2cl-lib:array-slice u
                              double-float
                              (1 i)
                              ((1 ldu) (1 *)))
        1))
(setf (f2cl-lib:fref q-%data% (1 i) ((1 ldq) (1 *)) q-%offset%)
      (/
        (f2cl-lib:fref u-%data% (1 i) ((1 ldu) (1 *)) u-%offset%)
        temp))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
              (> j k) nil)
(tagbody
  (setf jc (f2cl-lib:fref idxc-%data% (j) ((1 *)) idxc-%offset%))
  (setf (f2cl-lib:fref q-%data% (j i) ((1 ldq) (1 *)) q-%offset%)
        (/
          (f2cl-lib:fref u-%data%
                        (jc i)
                        ((1 ldu) (1 *))
                        u-%offset%)
          temp))))))
(cond
  ((= k 2)
   (dgemm "N" "N" n k k one u2 ldu2 q ldq zero u ldu)
   (go label100)))
(cond
  (> (f2cl-lib:fref ctot (1) ((1 *))) 0)
  (dgemm "N" "N" nl k
    (f2cl-lib:fref ctot-%data% (1) ((1 *)) ctot-%offset%) one

```

```

(f2cl-lib:array-slice u2 double-float (1 2) ((1 ldu2) (1 *))) ldu2
(f2cl-lib:array-slice q double-float (2 1) ((1 ldq) (1 *))) ldq
zero (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
ldu)
(cond
  (> (f2cl-lib:fref ctot (3) ((1 *))) 0)
  (setf ktemp
    (f2cl-lib:int-add 2
      (f2cl-lib:fref ctot-%data%
        (1)
        ((1 *))
        ctot-%offset%)
      (f2cl-lib:fref ctot-%data%
        (2)
        ((1 *))
        ctot-%offset%)))
  (dgemm "N" "N" nl k
    (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
    (f2cl-lib:array-slice u2
      double-float
      (1 ktemp)
      ((1 ldu2) (1 *)))
    ldu2
    (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *)))
    ldq one
    (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
    ldu)))
(> (f2cl-lib:fref ctot (3) ((1 *))) 0)
(setf ktemp
  (f2cl-lib:int-add 2
    (f2cl-lib:fref ctot-%data%
      (1)
      ((1 *))
      ctot-%offset%)
    (f2cl-lib:fref ctot-%data%
      (2)
      ((1 *))
      ctot-%offset%)))
(dgemm "N" "N" nl k
  (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
  (f2cl-lib:array-slice u2 double-float (1 ktemp) ((1 ldu2) (1 *)))
  ldu2
  (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *))) ldq
  zero (f2cl-lib:array-slice u double-float (1 1) ((1 ldu) (1 *)))
  ldu))
(t
  (dlacpy "F" nl k u2 ldu2 u ldu)))
(dcopy k (f2cl-lib:array-slice q double-float (1 1) ((1 ldq) (1 *)))
  ldq (f2cl-lib:array-slice u double-float (nlp1 1) ((1 ldu) (1 *)))
  ldu)

```

```

(setf ktemp
  (f2cl-lib:int-add 2
    (f2cl-lib:fref ctot-%data%
      (1)
      ((1 *))
      ctot-%offset%)))

(setf ctemp
  (f2cl-lib:int-add
    (f2cl-lib:fref ctot-%data% (2) ((1 *)) ctot-%offset%)
    (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%)))

(dgemm "N" "N" nr k ctemp one
  (f2cl-lib:array-slice u2 double-float (nlp2 ktemp) ((1 ldu2) (1 *)))
  ldu2 (f2cl-lib:array-slice q double-float (ktemp 1) ((1 ldq) (1 *)))
  ldq zero
  (f2cl-lib:array-slice u double-float (nlp2 1) ((1 ldu) (1 *))) ldu)

label100
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i k) nil)

(tagbody
  (setf temp
    (dnrm2 k
      (f2cl-lib:array-slice vt
        double-float
        (1 i)
        ((1 ldvt) (1 *)))
      1))
  (setf (f2cl-lib:fref q-%data% (i 1) ((1 ldq) (1 *)) q-%offset%)
    (/
      (f2cl-lib:fref vt-%data%
        (1 i)
        ((1 ldvt) (1 *))
        vt-%offset%)
      temp))
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
    (> j k) nil)
  (tagbody
    (setf jc (f2cl-lib:fref idxc-%data% (j) ((1 *)) idxc-%offset%))
    (setf (f2cl-lib:fref q-%data% (i j) ((1 ldq) (1 *)) q-%offset%)
      (/
        (f2cl-lib:fref vt-%data%
          (jc i)
          ((1 ldvt) (1 *))
          vt-%offset%)
        temp))))))

(cond
  (= k 2)
  (dgemm "N" "N" k m k one q ldq vt2 ldvt2 zero vt ldvt)
  (go end_label)))

(setf ktemp
  (f2cl-lib:int-add 1

```

```

(f2cl-lib:fref ctot-%data%
  (1)
  ((1 *))
  ctot-%offset%)))

(dgemm "N" "N" k nlp1 ktemp one
  (f2cl-lib:array-slice q double-float (1 1) ((1 ldq) (1 *))) ldq
  (f2cl-lib:array-slice vt2 double-float (1 1) ((1 ldvt2) (1 *))) ldvt2
  zero (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
  ldvt)
(setf ktemp
  (f2cl-lib:int-add 2
    (f2cl-lib:fref ctot-%data%
      (1)
      ((1 *))
      ctot-%offset%)
    (f2cl-lib:fref ctot-%data%
      (2)
      ((1 *))
      ctot-%offset%)))

(if (<= ktemp ldvt2)
  (dgemm "N" "N" k nlp1
    (f2cl-lib:fref ctot-%data% (3) ((1 *)) ctot-%offset%) one
    (f2cl-lib:array-slice q double-float (1 ktemp) ((1 ldq) (1 *)))
    ldq
    (f2cl-lib:array-slice vt2
      double-float
      (ktemp 1)
      ((1 ldvt2) (1 *)))

    ldvt2 one
    (f2cl-lib:array-slice vt double-float (1 1) ((1 ldvt) (1 *)))
    ldvt))
(setf ktemp
  (f2cl-lib:int-add
    (f2cl-lib:fref ctot-%data% (1) ((1 *)) ctot-%offset%)
    1))
(setf nrp1 (f2cl-lib:int-add nr sqre))
(cond
  (> ktemp 1)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i k) nil)
  (tagbody
    (setf (f2cl-lib:fref q-%data%
      (i ktemp)
      ((1 ldq) (1 *))
      q-%offset%)
      (f2cl-lib:fref q-%data%
        (i 1)
        ((1 ldq) (1 *))
        q-%offset%)))
    (f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))

```

dlasd4 LAPACK**— dlasd4.input —**

```

)set break resume
)sys rm -f dlasd4.output
)spool dlasd4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd4.help —

```

=====
dlasd4 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASD4 - compute the square root of the I-th updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix whose entries are given as the squares of the corresponding entries in the array d, and that $0 \leq D(i) < D(j)$ for $i < j$ and that $RHO > 0$

SYNOPSIS

```
SUBROUTINE DLASD4( N, I, D, Z, DELTA, RHO, SIGMA, WORK, INFO )
```

```
INTEGER          I, INFO, N
```

```
DOUBLE           PRECISION RHO, SIGMA
```

```
DOUBLE           PRECISION D( * ), DELTA( * ), WORK( * ), Z( * )
```

Purpose

```
=====
```

This subroutine computes the square root of the I-th updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix whose entries are given as the squares of the corresponding entries in the array d, and that

$$0 \leq D(i) < D(j) \quad \text{for } i < j$$

and that $RHO > 0$. This is arranged by the calling routine, and is no loss in generality. The rank-one modified system is thus

$$\text{diag}(D) * \text{diag}(D) + RHO * Z * Z_{\text{transpose}}.$$

where we assume the Euclidean norm of Z is 1.

The method consists of approximating the rational functions in the secular equation by simpler interpolating rational functions.

Arguments

=====

- N** (input) INTEGER
The length of all arrays.
- I** (input) INTEGER
The index of the eigenvalue to be computed. $1 \leq I \leq N$.
- D** (input) DOUBLE PRECISION array, dimension (N)
The original eigenvalues. It is assumed that they are in order, $0 \leq D(I) < D(J)$ for $I < J$.
- Z** (input) DOUBLE PRECISION array, dimension (N)
The components of the updating vector.
- DELTA** (output) DOUBLE PRECISION array, dimension (N)
If $N \neq 1$, DELTA contains $(D(j) - \text{sigma}_I)$ in its j -th component. If $N = 1$, then $DELTA(1) = 1$. The vector DELTA contains the information necessary to construct the (singular) eigenvectors.
- RHO** (input) DOUBLE PRECISION
The scalar in the symmetric updating formula.
- SIGMA** (output) DOUBLE PRECISION
The computed λ_I , the I -th updated eigenvalue.
- WORK** (workspace) DOUBLE PRECISION array, dimension (N)
If $N \neq 1$, WORK contains $(D(j) + \text{sigma}_I)$ in its j -th component. If $N = 1$, then $WORK(1) = 1$.
- INFO** (output) INTEGER
= 0: successful exit
> 0: if INFO = 1, the updating process failed.

Internal Parameters

=====

Logical variable ORGATI (origin-at-i?) is used for distinguishing whether D(i) or D(i+1) is treated as the origin.

```

      ORGATI = .true.    origin at i
      ORGATI = .false.   origin at i+1

```

Logical variable SWTCH3 (switch-for-3-poles?) is for noting if we are working with THREE poles!

MAXIT is the maximum number of iterations allowed for each eigenvalue.

Further Details
=====

Based on contributions by
Ren-Cang Li, Computer Science Division, University of California
at Berkeley, USA

— dlasd4.f —

```

      SUBROUTINE DLASD4( N, I, D, Z, DELTA, RHO, SIGMA, WORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*    Courant Institute, NAG Ltd., and Rice University
*    October 31, 1999
*
*    .. Scalar Arguments ..
*      INTEGER          I, INFO, N
*      DOUBLE PRECISION RHO, SIGMA
*
*    ..
*
*    .. Array Arguments ..
*      DOUBLE PRECISION D( * ), DELTA( * ), WORK( * ), Z( * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
*      INTEGER          MAXIT
*      PARAMETER        ( MAXIT = 20 )
*      DOUBLE PRECISION ZERO, ONE, TWO, THREE, FOUR, EIGHT, TEN
*      PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0,
*      $                 THREE = 3.0D+0, FOUR = 4.0D+0, EIGHT = 8.0D+0,

```

```

$          TEN = 10.0D+0 )
*
*      ..
*      .. Local Scalars ..
LOGICAL      ORGATI, SWTCH, SWTCH3
INTEGER      II, IIM1, IIP1, IP1, ITER, J, NITER
DOUBLE PRECISION  A, B, C, DELSQ, DELSQ2, DPHI, DPSI, DTIIM,
$            DTIIP, DTIPSQ, DTISQ, DTNSQ, DTNSQ1, DW, EPS,
$            ERRETM, ETA, PHI, PREW, PSI, RHOINV, SG2LB,
$            SG2UB, TAU, TEMP, TEMP1, TEMP2, W
*
*      ..
*      .. Local Arrays ..
DOUBLE PRECISION  DD( 3 ), ZZ( 3 )
*
*      ..
*      .. External Subroutines ..
EXTERNAL        DLAED6, DLASD5
*
*      ..
*      .. External Functions ..
DOUBLE PRECISION  DLAMCH
EXTERNAL        DLAMCH
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC        ABS, MAX, MIN, SQRT
*
*      ..
*      .. Executable Statements ..
*
*      Since this routine is called in an inner loop, we do no argument
*      checking.
*
*      Quick return for N=1 and 2.
*
*
INFO = 0
IF( N.EQ.1 ) THEN
*
*      Presumably, I=1 upon entry
*
*
SIGMA = SQRT( D( 1 )*D( 1 )+RHO*Z( 1 )*Z( 1 ) )
DELTA( 1 ) = ONE
WORK( 1 ) = ONE
RETURN
END IF
IF( N.EQ.2 ) THEN
CALL DLASD5( I, D, Z, DELTA, RHO, SIGMA, WORK )
RETURN
END IF
*
*      Compute machine epsilon
*
*
EPS = DLAMCH( 'Epsilon' )
RHOINV = ONE / RHO
*

```

```

*      The case I = N
*
*      IF( I.EQ.N ) THEN
*
*          Initialize some basic variables
*
*          II = N - 1
*          NITER = 1
*
*          Calculate initial guess
*
*          TEMP = RHO / TWO
*
*          If ||Z||_2 is not one, then TEMP should be set to
*          RHO * ||Z||_2^2 / TWO
*
*          TEMP1 = TEMP / ( D( N )+SQRT( D( N )*D( N )+TEMP ) )
*          DO 10 J = 1, N
*              WORK( J ) = D( J ) + D( N ) + TEMP1
*              DELTA( J ) = ( D( J )-D( N ) ) - TEMP1
10      CONTINUE
*
*          PSI = ZERO
*          DO 20 J = 1, N - 2
*              PSI = PSI + Z( J )*Z( J ) / ( DELTA( J )*WORK( J ) )
20      CONTINUE
*
*          C = RHOINV + PSI
*          W = C + Z( II )*Z( II ) / ( DELTA( II )*WORK( II ) ) +
$          Z( N )*Z( N ) / ( DELTA( N )*WORK( N ) )
*
*          IF( W.LE.ZERO ) THEN
*              TEMP1 = SQRT( D( N )*D( N )+RHO )
*              TEMP = Z( N-1 )*Z( N-1 ) / ( ( D( N-1 )+TEMP1 ) *
$              ( D( N )-D( N-1 )+RHO / ( D( N )+TEMP1 ) ) ) +
$              Z( N )*Z( N ) / RHO
*
*          The following TAU is to approximate
*          SIGMA_n^2 - D( N )*D( N )
*
*          IF( C.LE.TEMP ) THEN
*              TAU = RHO
*          ELSE
*              DELSQ = ( D( N )-D( N-1 ) )*( D( N )+D( N-1 ) )
*              A = -C*DELSQ + Z( N-1 )*Z( N-1 ) + Z( N )*Z( N )
*              B = Z( N )*Z( N )*DELSQ
*              IF( A.LT.ZERO ) THEN
*                  TAU = TWO*B / ( SQRT( A*A+FOUR*B*C )-A )
*              ELSE
*                  TAU = ( A+SQRT( A*A+FOUR*B*C ) ) / ( TWO*C )

```

```

        END IF
    END IF

*
*      It can be proved that
*       $D(N)^{2+\text{RHO}/2} \leq \text{SIGMA}_n^2 < D(N)^{2+\text{TAU}} \leq D(N)^{2+\text{RHO}}$ 
*
    ELSE
        DELSQ = ( D( N )-D( N-1 ) )*( D( N )+D( N-1 ) )
        A = -C*DELSQ + Z( N-1 )*Z( N-1 ) + Z( N )*Z( N )
        B = Z( N )*Z( N )*DELSQ

*
*      The following TAU is to approximate
*       $\text{SIGMA}_n^2 - D( N ) * D( N )$ 
*
        IF( A.LT.ZERO ) THEN
            TAU = TWO*B / ( SQRT( A*A+FOUR*B*C )-A )
        ELSE
            TAU = ( A+SQRT( A*A+FOUR*B*C ) ) / ( TWO*C )
        END IF

*
*      It can be proved that
*       $D(N)^2 < D(N)^{2+\text{TAU}} < \text{SIGMA}(N)^2 < D(N)^{2+\text{RHO}/2}$ 
*
    END IF

*
*      The following ETA is to approximate  $\text{SIGMA}_n - D( N )$ 
*
        ETA = TAU / ( D( N )+SQRT( D( N )*D( N )+TAU ) )

*
        SIGMA = D( N ) + ETA
        DO 30 J = 1, N
            DELTA( J ) = ( D( J )-D( I ) ) - ETA
            WORK( J ) = D( J ) + D( I ) + ETA
30      CONTINUE

*
*      Evaluate PSI and the derivative DPSI
*
        DPSI = ZERO
        PSI = ZERO
        ERRETM = ZERO
        DO 40 J = 1, II
            TEMP = Z( J ) / ( DELTA( J )*WORK( J ) )
            PSI = PSI + Z( J )*TEMP
            DPSI = DPSI + TEMP*TEMP
            ERRETM = ERRETM + PSI
40      CONTINUE
        ERRETM = ABS( ERRETM )

*
*      Evaluate PHI and the derivative DPHI
*

```

```

TEMP = Z( N ) / ( DELTA( N )*WORK( N ) )
PHI = Z( N )*TEMP
DPHI = TEMP*TEMP
ERRETM = EIGHT*( -PHI-PSI ) + ERRETM - PHI + RHOINV +
$      ABS( TAU )*( DPSI+DPHI )
*
W = RHOINV + PHI + PSI
*
*      Test for convergence
*
IF( ABS( W ).LE.EPS*ERRETM ) THEN
    GO TO 240
END IF
*
*      Calculate the new step
*
NITER = NITER + 1
DTNSQ1 = WORK( N-1 )*DELTA( N-1 )
DTNSQ = WORK( N )*DELTA( N )
C = W - DTNSQ1*DPSI - DTNSQ*DPHI
A = ( DTNSQ+DTNSQ1 )*W - DTNSQ*DTNSQ1*( DPSI+DPHI )
B = DTNSQ*DTNSQ1*W
IF( C.LT.ZERO )
$      C = ABS( C )
IF( C.EQ.ZERO ) THEN
    ETA = RHO - SIGMA*SIGMA
ELSE IF( A.GE.ZERO ) THEN
    ETA = ( A+SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
ELSE
    ETA = TWO*B / ( A-SQRT( ABS( A*A-FOUR*B*C ) ) )
END IF
*
*      Note, eta should be positive if w is negative, and
*      eta should be negative otherwise. However,
*      if for some reason caused by roundoff, eta*w > 0,
*      we simply use one Newton step instead. This way
*      will guarantee eta*w < 0.
*
IF( W*ETA.GT.ZERO )
$      ETA = -W / ( DPSI+DPHI )
TEMP = ETA - DTNSQ
IF( TEMP.GT.RHO )
$      ETA = RHO + DTNSQ
*
TAU = TAU + ETA
ETA = ETA / ( SIGMA+SQRT( ETA+SIGMA*SIGMA ) )
DO 50 J = 1, N
    DELTA( J ) = DELTA( J ) - ETA
    WORK( J ) = WORK( J ) + ETA
50 CONTINUE

```



```

*
      SIGMA = SIGMA + ETA
*
*   Evaluate PSI and the derivative DPSI
*
      DPSI = ZERO
      PSI = ZERO
      ERRETM = ZERO
      DO 60 J = 1, II
          TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
          PSI = PSI + Z( J )*TEMP
          DPSI = DPSI + TEMP*TEMP
          ERRETM = ERRETM + PSI
60    CONTINUE
      ERRETM = ABS( ERRETM )
*
*   Evaluate PHI and the derivative DPHI
*
      TEMP = Z( N ) / ( WORK( N )*DELTA( N ) )
      PHI = Z( N )*TEMP
      DPHI = TEMP*TEMP
      ERRETM = EIGHT*( -PHI-PSI ) + ERRETM - PHI + RHOINV +
$      ABS( TAU )*( DPSI+DPHI )
*
      W = RHOINV + PHI + PSI
*
*   Main loop to update the values of the array   DELTA
*
      ITER = NITER + 1
*
      DO 90 NITER = ITER, MAXIT
*
*       Test for convergence
*
          IF( ABS( W ).LE.EPS*ERRETM ) THEN
              GO TO 240
          END IF
*
*       Calculate the new step
*
          DTNSQ1 = WORK( N-1 )*DELTA( N-1 )
          DTNSQ = WORK( N )*DELTA( N )
          C = W - DTNSQ1*DPSI - DTNSQ*DPHI
          A = ( DTNSQ+DTNSQ1 )*W - DTNSQ1*DTNSQ*( DPSI+DPHI )
          B = DTNSQ1*DTNSQ*W
          IF( A.GE.ZERO ) THEN
              ETA = ( A+SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
          ELSE
              ETA = TWO*B / ( A-SQRT( ABS( A*A-FOUR*B*C ) ) )
          END IF

```

```

*
*      Note, eta should be positive if w is negative, and
*      eta should be negative otherwise. However,
*      if for some reason caused by roundoff, eta*w > 0,
*      we simply use one Newton step instead. This way
*      will guarantee eta*w < 0.
*
      IF( W*ETA.GT.ZERO )
$         ETA = -W / ( DPSI+DPHI )
      TEMP = ETA - DTNSQ
      IF( TEMP.LE.ZERO )
$         ETA = ETA / TWO
*
      TAU = TAU + ETA
      ETA = ETA / ( SIGMA+SQRT( ETA+SIGMA*SIGMA ) )
      DO 70 J = 1, N
          DELTA( J ) = DELTA( J ) - ETA
          WORK( J ) = WORK( J ) + ETA
70      CONTINUE
*
      SIGMA = SIGMA + ETA
*
*      Evaluate PSI and the derivative DPSI
*
      DPSI = ZERO
      PSI = ZERO
      ERRETM = ZERO
      DO 80 J = 1, II
          TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
          PSI = PSI + Z( J )*TEMP
          DPSI = DPSI + TEMP*TEMP
          ERRETM = ERRETM + PSI
80      CONTINUE
      ERRETM = ABS( ERRETM )
*
*      Evaluate PHI and the derivative DPHI
*
      TEMP = Z( N ) / ( WORK( N )*DELTA( N ) )
      PHI = Z( N )*TEMP
      DPHI = TEMP*TEMP
      ERRETM = EIGHT*( -PHI-PSI ) + ERRETM - PHI + RHOINV +
$          ABS( TAU )*( DPSI+DPHI )
*
      W = RHOINV + PHI + PSI
90      CONTINUE
*
*      Return with INFO = 1, NITER = MAXIT and not converged
*
      INFO = 1
      GO TO 240

```

```

*
*      End for the case I = N
*
*      ELSE
*
*      The case for I < N
*
*      NITER = 1
*      IP1 = I + 1
*
*      Calculate initial guess
*
      DELSQ = ( D( IP1 )-D( I ) )*( D( IP1 )+D( I ) )
      DELSQ2 = DELSQ / TWO
      TEMP = DELSQ2 / ( D( I )+SQRT( D( I )*D( I )+DELSQ2 ) )
      DO 100 J = 1, N
        WORK( J ) = D( J ) + D( I ) + TEMP
        DELTA( J ) = ( D( J )-D( I ) ) - TEMP
100    CONTINUE
*
      PSI = ZERO
      DO 110 J = 1, I - 1
        PSI = PSI + Z( J )*Z( J ) / ( WORK( J )*DELTA( J ) )
110    CONTINUE
*
      PHI = ZERO
      DO 120 J = N, I + 2, -1
        PHI = PHI + Z( J )*Z( J ) / ( WORK( J )*DELTA( J ) )
120    CONTINUE
      C = RHOINV + PSI + PHI
      W = C + Z( I )*Z( I ) / ( WORK( I )*DELTA( I ) ) +
$      Z( IP1 )*Z( IP1 ) / ( WORK( IP1 )*DELTA( IP1 ) )
*
      IF( W.GT.ZERO ) THEN
*
*       $d(i)^2 < \text{the } i\text{th } \sigma^2 < (d(i)^2 + d(i+1)^2)/2$ 
*
*      We choose d(i) as origin.
*
      ORGATI = .TRUE.
      SG2LB = ZERO
      SG2UB = DELSQ2
      A = C*DELSQ + Z( I )*Z( I ) + Z( IP1 )*Z( IP1 )
      B = Z( I )*Z( I )*DELSQ
      IF( A.GT.ZERO ) THEN
        TAU = TWO*B / ( A+SQRT( ABS( A*A-FOUR*B*C ) ) )
      ELSE
        TAU = ( A-SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
      END IF
*

```

```

*      TAU now is an estimation of  $\text{SIGMA}^2 - D(I)^2$ . The
*      following, however, is the corresponding estimation of
*       $\text{SIGMA} - D(I)$ .
*
      ETA = TAU / ( D( I )+SQRT( D( I )*D( I )+TAU ) )
ELSE
*
      (d(i)^2+d(i+1)^2)/2 <= the ith sigma^2 < d(i+1)^2/2
*
      We choose d(i+1) as origin.
*
      ORGATI = .FALSE.
      SG2LB = -DELSQ2
      SG2UB = ZERO
      A = C*DELSQ - Z( I )*Z( I ) - Z( IP1 )*Z( IP1 )
      B = Z( IP1 )*Z( IP1 )*DELSQ
      IF( A.LT.ZERO ) THEN
          TAU = TWO*B / ( A-SQRT( ABS( A*A+FOUR*B*C ) ) )
      ELSE
          TAU = -( A+SQRT( ABS( A*A+FOUR*B*C ) ) ) / ( TWO*C )
      END IF
*
      TAU now is an estimation of  $\text{SIGMA}^2 - D(IP1)^2$ . The
      following, however, is the corresponding estimation of
       $\text{SIGMA} - D(IP1)$ .
*
      ETA = TAU / ( D( IP1 )+SQRT( ABS( D( IP1 )*D( IP1 )+
$      TAU ) ) )
      END IF
*
      IF( ORGATI ) THEN
          II = I
          SIGMA = D( I ) + ETA
          DO 130 J = 1, N
              WORK( J ) = D( J ) + D( I ) + ETA
              DELTA( J ) = ( D( J )-D( I ) ) - ETA
130          CONTINUE
      ELSE
          II = I + 1
          SIGMA = D( IP1 ) + ETA
          DO 140 J = 1, N
              WORK( J ) = D( J ) + D( IP1 ) + ETA
              DELTA( J ) = ( D( J )-D( IP1 ) ) - ETA
140          CONTINUE
      END IF
      IIM1 = II - 1
      IIP1 = II + 1
*
*      Evaluate PSI and the derivative DPSI
*

```

```

    DPSI = ZERO
    PSI = ZERO
    ERRETM = ZERO
    DO 150 J = 1, IIM1
        TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
        PSI = PSI + Z( J )*TEMP
        DPSI = DPSI + TEMP*TEMP
        ERRETM = ERRETM + PSI
150    CONTINUE
    ERRETM = ABS( ERRETM )
*
*    Evaluate PHI and the derivative DPHI
*
    DPHI = ZERO
    PHI = ZERO
    DO 160 J = N, IIP1, -1
        TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
        PHI = PHI + Z( J )*TEMP
        DPHI = DPHI + TEMP*TEMP
        ERRETM = ERRETM + PHI
160    CONTINUE
*
*    W = RHOINV + PHI + PSI
*
*    W is the value of the secular function with
*    its ii-th element removed.
*
    SWTCH3 = .FALSE.
    IF( ORGATI ) THEN
        IF( W.LT.ZERO )
$           SWTCH3 = .TRUE.
    ELSE
        IF( W.GT.ZERO )
$           SWTCH3 = .TRUE.
    END IF
    IF( II.EQ.1 .OR. II.EQ.N )
$       SWTCH3 = .FALSE.
*
    TEMP = Z( II ) / ( WORK( II )*DELTA( II ) )
    DW = DPSI + DPHI + TEMP*TEMP
    TEMP = Z( II )*TEMP
    W = W + TEMP
    ERRETM = EIGHT*( PHI-PSI ) + ERRETM + TWO*RHOINV +
$       THREE*ABS( TEMP ) + ABS( TAU )*DW
*
*    Test for convergence
*
    IF( ABS( W ).LE.EPS*ERRETM ) THEN
        GO TO 240
    END IF

```

```

*
IF( W.LE.ZERO ) THEN
    SG2LB = MAX( SG2LB, TAU )
ELSE
    SG2UB = MIN( SG2UB, TAU )
END IF

*
* Calculate the new step
*
NITER = NITER + 1
IF( .NOT.SWTC3 ) THEN
    DTIPSQ = WORK( IP1 )*DELTA( IP1 )
    DTISQ = WORK( I )*DELTA( I )
    IF( ORGATI ) THEN
        C = W - DTIPSQ*DW + DELSQ*( Z( I ) / DTISQ )**2
    ELSE
        C = W - DTISQ*DW - DELSQ*( Z( IP1 ) / DTIPSQ )**2
    END IF
    A = ( DTIPSQ+DTISQ )*W - DTIPSQ*DTISQ*DW
    B = DTIPSQ*DTISQ*W
    IF( C.EQ.ZERO ) THEN
        IF( A.EQ.ZERO ) THEN
            IF( ORGATI ) THEN
                A = Z( I )*Z( I ) + DTIPSQ*DTIPSQ*( DPSI+DPHI )
            ELSE
                A = Z( IP1 )*Z( IP1 ) + DTISQ*DTISQ*( DPSI+DPHI )
            END IF
        END IF
        ETA = B / A
    ELSE IF( A.LE.ZERO ) THEN
        ETA = ( A-SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
    ELSE
        ETA = TWO*B / ( A+SQRT( ABS( A*A-FOUR*B*C ) ) )
    END IF
ELSE
    Interpolation using THREE most relevant poles

    DTIIM = WORK( IIM1 )*DELTA( IIM1 )
    DTIIP = WORK( IIP1 )*DELTA( IIP1 )
    TEMP = RHOINV + PSI + PHI
    IF( ORGATI ) THEN
        TEMP1 = Z( IIM1 ) / DTIIM
        TEMP1 = TEMP1*TEMP1
        C = ( TEMP - DTIIP*( DPSI+DPHI ) ) -
$      ( D( IIM1 )-D( IIP1 ) )*( D( IIM1 )+D( IIP1 ) )*TEMP1
        ZZ( 1 ) = Z( IIM1 )*Z( IIM1 )
        IF( DPSI.LT.TEMP1 ) THEN
            ZZ( 3 ) = DTIIP*DTIIP*DPHI
        ELSE

```

```

        ZZ( 3 ) = DTIIP*DTIIP*( ( DPSI-TEMP1 )+DPHI )
      END IF
    ELSE
      TEMP1 = Z( IIP1 ) / DTIIP
      TEMP1 = TEMP1*TEMP1
      C = ( TEMP - DTIIM*( DPSI+DPHI ) ) -
$      ( D( IIP1 )-D( IIM1 ) )*( D( IIM1 )+D( IIP1 ) )*TEMP1
      IF( DPHI.LT.TEMP1 ) THEN
        ZZ( 1 ) = DTIIM*DTIIM*DPSI
      ELSE
        ZZ( 1 ) = DTIIM*DTIIM*( DPSI+( DPHI-TEMP1 ) )
      END IF
      ZZ( 3 ) = Z( IIP1 )*Z( IIP1 )
    END IF
    ZZ( 2 ) = Z( II )*Z( II )
    DD( 1 ) = DTIIM
    DD( 2 ) = DELTA( II )*WORK( II )
    DD( 3 ) = DTIIP
    CALL DLAED6( NITER, ORGATI, C, DD, ZZ, W, ETA, INFO )
    IF( INFO.NE.0 )
$      GO TO 240
    END IF

*
*   Note, eta should be positive if w is negative, and
*   eta should be negative otherwise. However,
*   if for some reason caused by roundoff, eta*w > 0,
*   we simply use one Newton step instead. This way
*   will guarantee eta*w < 0.
*
    IF( W*ETA.GE.ZERO )
$      ETA = -W / DW
    IF( ORGATI ) THEN
      TEMP1 = WORK( I )*DELTA( I )
      TEMP = ETA - TEMP1
    ELSE
      TEMP1 = WORK( IP1 )*DELTA( IP1 )
      TEMP = ETA - TEMP1
    END IF
    IF( TEMP.GT.SG2UB .OR. TEMP.LT.SG2LB ) THEN
      IF( W.LT.ZERO ) THEN
        ETA = ( SG2UB-TAU ) / TWO
      ELSE
        ETA = ( SG2LB-TAU ) / TWO
      END IF
    END IF

*
    TAU = TAU + ETA
    ETA = ETA / ( SIGMA+SQRT( SIGMA*SIGMA+ETA ) )

*
    PREW = W

```

```

*
      SIGMA = SIGMA + ETA
      DO 170 J = 1, N
          WORK( J ) = WORK( J ) + ETA
          DELTA( J ) = DELTA( J ) - ETA
170      CONTINUE
*
*      Evaluate PSI and the derivative DPSI
*
      DPSI = ZERO
      PSI = ZERO
      ERRETM = ZERO
      DO 180 J = 1, IIM1
          TEMP = Z( J ) / ( WORK( J ) * DELTA( J ) )
          PSI = PSI + Z( J ) * TEMP
          DPSI = DPSI + TEMP * TEMP
          ERRETM = ERRETM + PSI
180      CONTINUE
      ERRETM = ABS( ERRETM )
*
*      Evaluate PHI and the derivative DPHI
*
      DPHI = ZERO
      PHI = ZERO
      DO 190 J = N, IIP1, -1
          TEMP = Z( J ) / ( WORK( J ) * DELTA( J ) )
          PHI = PHI + Z( J ) * TEMP
          DPHI = DPHI + TEMP * TEMP
          ERRETM = ERRETM + PHI
190      CONTINUE
*
      TEMP = Z( II ) / ( WORK( II ) * DELTA( II ) )
      DW = DPSI + DPHI + TEMP * TEMP
      TEMP = Z( II ) * TEMP
      W = RHOINV + PHI + PSI + TEMP
      ERRETM = EIGHT * ( PHI - PSI ) + ERRETM + TWO * RHOINV +
$          THREE * ABS( TEMP ) + ABS( TAU ) * DW
*
      IF( W.LE.ZERO ) THEN
          SG2LB = MAX( SG2LB, TAU )
      ELSE
          SG2UB = MIN( SG2UB, TAU )
      END IF
*
      SWITCH = .FALSE.
      IF( ORGATI ) THEN
          IF( -W.GT.ABS( PREW ) / TEN )
$              SWITCH = .TRUE.
      ELSE
          IF( W.GT.ABS( PREW ) / TEN )

```



```

$          SWITCH = .TRUE.
      END IF

*
*      Main loop to update the values of the array   DELTA and WORK
*
      ITER = NITER + 1
*
      DO 230 NITER = ITER, MAXIT
*
*          Test for convergence
*
          IF( ABS( W ).LE.EPS*ERRETM ) THEN
              GO TO 240
          END IF
*
*          Calculate the new step
*
          IF( .NOT.SWITCH3 ) THEN
              DTIPSQ = WORK( IP1 )*DELTA( IP1 )
              DTISQ = WORK( I )*DELTA( I )
              IF( .NOT.SWITCH ) THEN
                  IF( ORGATI ) THEN
                      C = W - DTIPSQ*DW + DELSQ*( Z( I ) / DTISQ )**2
                  ELSE
                      C = W - DTISQ*DW - DELSQ*( Z( IP1 ) / DTIPSQ )**2
                  END IF
              ELSE
                  TEMP = Z( II ) / ( WORK( II )*DELTA( II ) )
                  IF( ORGATI ) THEN
                      DPSI = DPSI + TEMP*TEMP
                  ELSE
                      DPHI = DPHI + TEMP*TEMP
                  END IF
                  C = W - DTISQ*DPSI - DTIPSQ*DPHI
              END IF
              A = ( DTIPSQ+DTISQ )*W - DTIPSQ*DTISQ*DW
              B = DTIPSQ*DTISQ*W
              IF( C.EQ.ZERO ) THEN
                  IF( A.EQ.ZERO ) THEN
                      IF( .NOT.SWITCH ) THEN
                          IF( ORGATI ) THEN
                              A = Z( I )*Z( I ) + DTIPSQ*DTIPSQ*
$                               ( DPSI+DPHI )
                          ELSE
                              A = Z( IP1 )*Z( IP1 ) +
$                               DTISQ*DTISQ*( DPSI+DPHI )
                          END IF
                      ELSE
                          A = DTISQ*DTISQ*DPSI + DTIPSQ*DTIPSQ*DPHI
                      END IF
                  END IF
              END IF
          END IF
      END DO

```

```

      END IF
      ETA = B / A
    ELSE IF( A.LE.ZERO ) THEN
      ETA = ( A-SQRT( ABS( A*A-FOUR*B*C ) ) ) / ( TWO*C )
    ELSE
      ETA = TWO*B / ( A+SQRT( ABS( A*A-FOUR*B*C ) ) )
    END IF
  ELSE
*
*      Interpolation using THREE most relevant poles
*
    DTIIM = WORK( IIM1 )*DELTA( IIM1 )
    DTIIP = WORK( IIP1 )*DELTA( IIP1 )
    TEMP = RHOINV + PSI + PHI
    IF( SWITCH ) THEN
      C = TEMP - DTIIM*DPSI - DTIIP*DPHI
      ZZ( 1 ) = DTIIM*DTIIM*DPSI
      ZZ( 3 ) = DTIIP*DTIIP*DPHI
    ELSE
      IF( ORGATI ) THEN
        TEMP1 = Z( IIM1 ) / DTIIM
        TEMP1 = TEMP1*TEMP1
        TEMP2 = ( D( IIM1 )-D( IIP1 ) ) *
          ( D( IIM1 )+D( IIP1 ) ) *TEMP1
        C = TEMP - DTIIP*( DPSI+DPHI ) - TEMP2
        ZZ( 1 ) = Z( IIM1 )*Z( IIM1 )
        IF( DPSI.LT.TEMP1 ) THEN
          ZZ( 3 ) = DTIIP*DTIIP*DPHI
        ELSE
          ZZ( 3 ) = DTIIP*DTIIP*( ( DPSI-TEMP1 )+DPHI )
        END IF
      ELSE
        TEMP1 = Z( IIP1 ) / DTIIP
        TEMP1 = TEMP1*TEMP1
        TEMP2 = ( D( IIP1 )-D( IIM1 ) ) *
          ( D( IIM1 )+D( IIP1 ) ) *TEMP1
        C = TEMP - DTIIM*( DPSI+DPHI ) - TEMP2
        IF( DPHI.LT.TEMP1 ) THEN
          ZZ( 1 ) = DTIIM*DTIIM*DPSI
        ELSE
          ZZ( 1 ) = DTIIM*DTIIM*( DPSI+( DPHI-TEMP1 ) )
        END IF
        ZZ( 3 ) = Z( IIP1 )*Z( IIP1 )
      END IF
    END IF
    DD( 1 ) = DTIIM
    DD( 2 ) = DELTA( II )*WORK( II )
    DD( 3 ) = DTIIP
    CALL DLAED6( NITER, ORGATI, C, DD, ZZ, W, ETA, INFO )
    IF( INFO.NE.0 )

```

```

$          GO TO 240
      END IF

*
*      Note, eta should be positive if w is negative, and
*      eta should be negative otherwise. However,
*      if for some reason caused by roundoff, eta*w > 0,
*      we simply use one Newton step instead. This way
*      will guarantee eta*w < 0.
*
      IF( W*ETA.GE.ZERO )
$      ETA = -W / DW
      IF( ORGATI ) THEN
        TEMP1 = WORK( I )*DELTA( I )
        TEMP = ETA - TEMP1
      ELSE
        TEMP1 = WORK( IP1 )*DELTA( IP1 )
        TEMP = ETA - TEMP1
      END IF
      IF( TEMP.GT.SG2UB .OR. TEMP.LT.SG2LB ) THEN
        IF( W.LT.ZERO ) THEN
          ETA = ( SG2UB-TAU ) / TWO
        ELSE
          ETA = ( SG2LB-TAU ) / TWO
        END IF
      END IF

*
      TAU = TAU + ETA
      ETA = ETA / ( SIGMA+SQRT( SIGMA*SIGMA+ETA ) )

*
      SIGMA = SIGMA + ETA
      DO 200 J = 1, N
        WORK( J ) = WORK( J ) + ETA
        DELTA( J ) = DELTA( J ) - ETA
200    CONTINUE

*
      PREW = W

*
*      Evaluate PSI and the derivative DPSI
*
      DPSI = ZERO
      PSI = ZERO
      ERRETM = ZERO
      DO 210 J = 1, IIM1
        TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
        PSI = PSI + Z( J )*TEMP
        DPSI = DPSI + TEMP*TEMP
        ERRETM = ERRETM + PSI
210    CONTINUE
      ERRETM = ABS( ERRETM )

*

```

```

*          Evaluate PHI and the derivative DPHI
*
      DPHI = ZERO
      PHI = ZERO
      DO 220 J = N, IIP1, -1
        TEMP = Z( J ) / ( WORK( J )*DELTA( J ) )
        PHI = PHI + Z( J )*TEMP
        DPHI = DPHI + TEMP*TEMP
        ERRETM = ERRETM + PHI
220      CONTINUE
*
      TEMP = Z( II ) / ( WORK( II )*DELTA( II ) )
      DW = DPSI + DPHI + TEMP*TEMP
      TEMP = Z( II )*TEMP
      W = RHOINV + PHI + PSI + TEMP
      ERRETM = EIGHT*( PHI-PSI ) + ERRETM + TWO*RHOINV +
$          THREE*ABS( TEMP ) + ABS( TAU )*DW
      IF( W*PREW.GT.ZERO .AND. ABS( W ).GT.ABS( PREW ) / TEN )
$          SWITCH = .NOT.SWITCH
*
      IF( W.LE.ZERO ) THEN
        SG2LB = MAX( SG2LB, TAU )
      ELSE
        SG2UB = MIN( SG2UB, TAU )
      END IF
*
230      CONTINUE
*
*          Return with INFO = 1, NITER = MAXIT and not converged
*
      INFO = 1
*
      END IF
*
240      CONTINUE
      RETURN
*
*          End of DLASD4
*
      END

```

— LAPACK dlasd4 —

```

(let* ((maxit 20)
      (zero 0.0)
      (one 1.0)

```

```

(two 2.0)
(three 3.0)
(four 4.0)
(eight 8.0)
(ten 10.0))
(declare (type (fixnum 20 20) maxit)
  (type (double-float 0.0 0.0) zero)
  (type (double-float 1.0 1.0) one)
  (type (double-float 2.0 2.0) two)
  (type (double-float 3.0 3.0) three)
  (type (double-float 4.0 4.0) four)
  (type (double-float 8.0 8.0) eight)
  (type (double-float 10.0 10.0) ten))
(defun dlasd4 (n i d z delta rho sigma work info)
  (declare (type (double-float) sigma rho)
    (type (simple-array double-float (*)) work delta z d)
    (type fixnum info i n))
  (f2cl-lib:with-multi-array-data
    ((d double-float d-%data% d-%offset%)
     (z double-float z-%data% z-%offset%)
     (delta double-float delta-%data% delta-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((dd (make-array 3 :element-type 'double-float))
      (zz (make-array 3 :element-type 'double-float)) (a 0.0) (b 0.0)
      (c 0.0) (delsq 0.0) (delsq2 0.0) (dphi 0.0) (dpsi 0.0) (dtiim 0.0)
      (dtiip 0.0) (dtipsq 0.0) (dtisq 0.0) (dtnsq 0.0) (dtnsq1 0.0)
      (dw 0.0) (eps 0.0) (erretm 0.0) (eta 0.0) (phi 0.0) (prew 0.0)
      (psi 0.0) (rhoinv 0.0) (sg2lb 0.0) (sg2ub 0.0) (tau 0.0)
      (temp 0.0) (temp1 0.0) (temp2 0.0) (w 0.0) (ii 0) (iim1 0)
      (iip1 0) (ip1 0) (iter 0) (j 0) (niter 0) (orgati nil) (swtch nil)
      (swtch3 nil))
      (declare (type (simple-array double-float (3)) dd zz)
        (type (double-float) a b c delsq delsq2 dphi dpsi dtiim dtiip
          dtipsq dtisq dtnsq dtnsq1 dw eps erretm
          eta phi prew psi rhoinv sg2lb sg2ub tau
          temp temp1 temp2 w)
        (type fixnum ii iim1 iip1 ip1 iter j niter)
        (type (member t nil) orgati swtch swtch3))
      (setf info 0)
      (cond
        ((= n 1)
          (setf sigma
            (f2cl-lib:fsqrt
              (+
                (* (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
                (* rho
                  (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
                  (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))))))
          (setf (f2cl-lib:fref delta-%data% (1) ((1 *)) delta-%offset%) one)

```

```

      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
      (go end_label)))
(cond
  ((= n 2)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
      (dlasd5 i d z delta rho sigma work)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6))
      (setf sigma var-5))
    (go end_label)))
(setf eps (dlamch "Epsilon"))
(setf rhoinv (/ one rho))
(cond
  ((= i n)
    (setf ii (f2cl-lib:int-sub n 1))
    (setf niter 1)
    (setf temp (/ rho two))
    (setf temp1
      (/ temp
        (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
          (f2cl-lib:fsqrt
            (+
              (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
                (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
            temp))))))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
          (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
            (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
            temp1))
        (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
          (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
            (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
            temp1))))
    (setf psi zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j (f2cl-lib:int-add n (f2cl-lib:int-sub 2))) nil)
      (tagbody
        (setf psi
          (+ psi
            (/
              (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
                (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
              (*
                (f2cl-lib:fref delta-%data%
                  (j)
                  ((1 *))
                  delta-%offset%)
                (f2cl-lib:fref work-%data%

```

```

                                (j)
                                ((1 *))
                                work-%offset%))))))
(setf c (+ rhoinv psi))
(setf w
  (+ c
    (/
      (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%))
      (*
        (f2cl-lib:fref delta-%data%
          (ii)
          ((1 *))
          delta-%offset%)
        (f2cl-lib:fref work-%data%
          (ii)
          ((1 *))
          work-%offset%)))
    (/
      (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%))
      (*
        (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)
        (f2cl-lib:fref work-%data%
          (n)
          ((1 *))
          work-%offset%))))))
(cond
  ((<= w zero)
   (setf temp1
     (f2cl-lib:fsqrt
      (+
        (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
           (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%))
        rho)))
   (setf temp
     (+
      (/
        (*
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub n 1))
            ((1 *))
            z-%offset%)
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub n 1))
            ((1 *))
            z-%offset%))
        (*
          (+
            (f2cl-lib:fref d-%data%
              ((f2cl-lib:int-sub n 1))
              ((1 *))
              d-%offset%)
            rho)))
      temp1)))

```

```

((f2cl-lib:int-sub n 1))
((1 *))
d-%offset%)

temp1)
(+
  (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
    (f2cl-lib:fref d-%data%
      ((f2cl-lib:int-sub n 1))
      ((1 *))
      d-%offset%))
  (/ rho
    (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
      temp1))))
(/
  (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    rho)))
(cond
  ((<= c temp)
    (setf tau rho))
  (t
    (setf delsq
      (*
        (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data%
            ((f2cl-lib:int-sub n 1))
            ((1 *))
            d-%offset%))
        (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data%
            ((f2cl-lib:int-sub n 1))
            ((1 *))
            d-%offset%))))
      (setf a
        (+ (* (- c) delsq)
          (*
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub n 1))
              ((1 *))
              z-%offset%)
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub n 1))
              ((1 *))
              z-%offset%))
          (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data%
              (n)
              ((1 *))
              z-%offset%))))))
    (setf b

```



```

      (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
         delsq))
(cond
  ((< a zero)
   (setf tau
    (/ (* two b)
        (- (f2cl-lib:fsqrt (+ (* a a) (* four b c)))
            a))))
  (t
   (setf tau
    (/ (+ a (f2cl-lib:fsqrt (+ (* a a) (* four b c))))
        (* two c))))))
(t
 (setf delsq
  (*
   (- (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
       (f2cl-lib:fref d-%data%
        ((f2cl-lib:int-sub n 1))
        ((1 *))
        d-%offset%))
   (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
       (f2cl-lib:fref d-%data%
        ((f2cl-lib:int-sub n 1))
        ((1 *))
        d-%offset%))))))
(setf a
 (+ (* (- c) delsq)
    (*
     (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub n 1))
      ((1 *))
      z-%offset%)
     (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub n 1))
      ((1 *))
      z-%offset%))
    (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%))))
(setf b
 (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    delsq))
(cond
  ((< a zero)
   (setf tau
    (/ (* two b)
        (- (f2cl-lib:fsqrt (+ (* a a) (* four b c))) a))))
  (t
   (setf tau

```

```

      (/ (+ a (f2cl-lib:fsqrt (+ (* a a) (* four b c))))
         (* two c))))))
(setf eta
  (/ tau
    (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
      (f2cl-lib:fsqrt
        (+
          (* (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
            (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
          tau))))))
(setf sigma (+ (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) eta))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
    (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
      (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
      eta))
  (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
    (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
      (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
      eta))))
(setf dpsl zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j ii) nil)
(tagbody
  (setf temp
    (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
      (*
        (f2cl-lib:fref delta-%data%
          (j)
          ((1 *))
          delta-%offset%)
        (f2cl-lib:fref work-%data%
          (j)
          ((1 *))
          work-%offset%))))))
  (setf psi
    (+ psi
      (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
        temp)))
  (setf dpsl (+ dpsl (* temp temp)))
  (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf temp
  (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    (*
      (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)

```

```

(f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)))
(setf phi (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%) temp))
(setf dphi (* temp temp))
(setf erretm
  (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
    rhoinv
    (* (abs tau) (+ dpsd dphi))))
(setf w (+ rhoinv phi psi))
(cond
  ((<= (abs w) (* eps erretm))
   (go end_label)))
(setf niter (f2cl-lib:int-add niter 1))
(setf dtnsq1
  (*
    (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-sub n 1))
      ((1 *))
      work-%offset%)
    (f2cl-lib:fref delta-%data%
      ((f2cl-lib:int-sub n 1))
      ((1 *))
      delta-%offset%)))
(setf dtnsq
  (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
    (f2cl-lib:fref delta-%data% (n) ((1 *)) delta-%offset%)))
(setf c (- w (* dtnsq1 dpsd) (* dtnsq dphi)))
(setf a
  (+ (* (+ dtnsq dtnsq1) w)
    (* (- dtnsq) dtnsq1 (+ dpsd dphi))))
(setf b (* dtnsq dtnsq1 w))
(if (< c zero) (setf c (abs c)))
(cond
  ((= c zero)
   (setf eta (- rho (* sigma sigma))))
  ((>= a zero)
   (setf eta
    (/
      (+ a
        (f2cl-lib:fsqrt (abs (+ (* a a) (* (- four) b c)))))
      (* two c))))
  (t
   (setf eta
    (/ (* two b)
      (- a
        (f2cl-lib:fsqrt
          (abs (+ (* a a) (* (- four) b c))))))))
(if (> (* w eta) zero) (setf eta (/ (- w) (+ dpsd dphi))))
(setf temp (- eta dtnsq))
(if (> temp rho) (setf eta (+ rho dtnsq)))
(setf tau (+ tau eta))

```

```

(setf eta (/ eta (+ sigma (f2cl-lib:fsqrt (+ eta (* sigma sigma))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
    (-
      (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
      eta))
  (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
    (+ (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
      eta))))
(setf sigma (+ sigma eta))
(setf dpsl zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j ii) nil)
(tagbody
  (setf temp
    (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
      (*
        (f2cl-lib:fref work-%data%
          (j)
          ((1 *))
          work-%offset%)
        (f2cl-lib:fref delta-%data%
          (j)
          ((1 *))
          delta-%offset%))))
  (setf psi
    (+ psi
      (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
        temp)))
  (setf dpsl (+ dpsl (* temp temp)))
  (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf temp
  (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
      (f2cl-lib:fref delta-%data%
        (n)
        ((1 *))
        delta-%offset%))))
(setf phi (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%) temp))
(setf dphi (* temp temp))
(setf erretm
  (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
    rhoinv
    (* (abs tau) (+ dpsl dphi))))
(setf w (+ rhoinv phi psi))

```

```

(setf iter (f2cl-lib:int-add niter 1))
(f2cl-lib:fd0 (niter iter (f2cl-lib:int-add niter 1))
  (> niter maxit) nil)
(tagbody
  (cond
    ((<= (abs w) (* eps erretm))
      (go end_label)))
  (setf dtnsq1
    (*
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-sub n 1))
        ((1 *))
        work-%offset%)
      (f2cl-lib:fref delta-%data%
        ((f2cl-lib:int-sub n 1))
        ((1 *))
        delta-%offset%)))
  (setf dtnsq
    (* (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%)
      (f2cl-lib:fref delta-%data%
        (n)
        ((1 *))
        delta-%offset%)))
  (setf c (- w (* dtnsq1 dpsi) (* dtnsq dphi)))
  (setf a
    (+ (* (+ dtnsq dtnsq1) w)
      (* (- dtnsq1) dtnsq (+ dpsi dphi))))
  (setf b (* dtnsq1 dtnsq w))
  (cond
    ((>= a zero)
      (setf eta
        (/
          (+ a
            (f2cl-lib:fsqrt
              (abs (+ (* a a) (* (- four) b c))))))
          (* two c))))
    (t
      (setf eta
        (/ (* two b)
          (- a
            (f2cl-lib:fsqrt
              (abs (+ (* a a) (* (- four) b c))))))))
    (if (> (* w eta) zero) (setf eta (/ (- w) (+ dpsi dphi))))
  (setf temp (- eta dtnsq))
  (if (<= temp zero) (setf eta (/ eta two)))
  (setf tau (+ tau eta))
  (setf eta
    (/ eta
      (+ sigma (f2cl-lib:fsqrt (+ eta (* sigma sigma))))))
  (f2cl-lib:fd0 (j 1 (f2cl-lib:int-add j 1))

```

```

        (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref delta-%data%
                      (j)
                      ((1 *))
                      delta-%offset%)
        (-
          (f2cl-lib:fref delta-%data%
                        (j)
                        ((1 *))
                        delta-%offset%)
          eta))
  (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (+
          (f2cl-lib:fref work-%data%
                        (j)
                        ((1 *))
                        work-%offset%)
          eta))))
(setf sigma (+ sigma eta))
(setf dpsi zero)
(setf psi zero)
(setf erretm zero)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j ii) nil)
(tagbody
  (setf temp
    (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
      (*
        (f2cl-lib:fref work-%data%
                      (j)
                      ((1 *))
                      work-%offset%)
        (f2cl-lib:fref delta-%data%
                      (j)
                      ((1 *))
                      delta-%offset%))))
  (setf psi
    (+ psi
      (*
        (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
        temp)))
  (setf dpsi (+ dpsi (* temp temp)))
  (setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf temp
  (/ (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    (*
      (f2cl-lib:fref work-%data%
                    (n)

```

```

                                ((1 *))
                                work-%offset%)
(f2cl-lib:fref delta-%data%
                                (n)
                                ((1 *))
                                delta-%offset%))))
(setf phi
  (* (f2cl-lib:fref z-%data% (n) ((1 *)) z-%offset%)
    temp))
(setf dphi (* temp temp))
(setf erretm
  (+ (- (+ (* eight (- (- psi) phi)) erretm) phi)
    rhoinv
    (* (abs tau) (+ dpsd dphi))))
(setf w (+ rhoinv phi psi)))
(setf info 1)
(go end_label))
(t
  (setf niter 1)
  (setf ip1 (f2cl-lib:int-add i 1))
  (setf delsq
    (*
      (- (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
      (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
    (setf delsq2 (/ delsq two))
    (setf temp
      (/ delsq2
        (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          (f2cl-lib:fsqrt
            (+
              (* (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
              delsq2))))))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j n) nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          temp))
      (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
        (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          temp))))
    (setf psi zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
    (tagbody

```

```

(setf psi
  (+ psi
    (/
      (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
      (*
        (f2cl-lib:fref work-%data%
          (j)
          ((1 *))
          work-%offset%)
        (f2cl-lib:fref delta-%data%
          (j)
          ((1 *))
          delta-%offset%))))))
(setf phi zero)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j (f2cl-lib:int-add i 2)) nil)
(tagbody
  (setf phi
    (+ phi
      (/
        (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
          (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
        (*
          (f2cl-lib:fref work-%data%
            (j)
            ((1 *))
            work-%offset%)
          (f2cl-lib:fref delta-%data%
            (j)
            ((1 *))
            delta-%offset%))))))
(setf c (+ rhoinv psi phi))
(setf w
  (+ c
    (/
      (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
      (* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (i)
          ((1 *))
          delta-%offset%)))
    (/
      (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))
      (*
        (f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (ip1)
          ((1 *))
          delta-%offset%))))))

```



```

                                ((1 *))
                                delta-%offset%))))))
(cond
  (> w zero)
  (setf orgati t)
  (setf sg2lb zero)
  (setf sg2ub delsq2)
  (setf a
    (+ (* c delsq)
      (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
      (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))))))
  (setf b
    (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
      delsq))
  (cond
    (> a zero)
    (setf tau
      (/ (* two b)
        (+ a
          (f2cl-lib:fsqrt
            (abs (+ (* a a) (* (- four) b c))))))))
    (t
      (setf tau
        (/
          (- a
            (f2cl-lib:fsqrt
              (abs (+ (* a a) (* (- four) b c)))))
          (* two c)))))
  (setf eta
    (/ tau
      (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fsqrt
          (+
            (*
              (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
              (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
              tau))))))
  (t
    (setf orgati nil)
    (setf sg2lb (- delsq2))
    (setf sg2ub zero)
    (setf a
      (- (* c delsq)
        (* (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
          (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))
        (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
          (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%))))))

```

```

(setf b
  (* (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
     (f2cl-lib:fref z-%data% (ip1) ((1 *)) z-%offset%)
     delsq))
(cond
  ((< a zero)
   (setf tau
     (/ (* two b)
        (- a
           (f2cl-lib:fsqrt
            (abs (+ (* a a) (* four b c))))))))
  (t
   (setf tau
     (/
      (-
       (+ a
          (f2cl-lib:fsqrt (abs (+ (* a a) (* four b c))))))
      (* two c))))
  (setf eta
    (/ tau
      (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
         (f2cl-lib:fsqrt
          (abs
           (+
            (*
             (f2cl-lib:fref d-%data%
                           (ip1)
                           ((1 *))
                           d-%offset%)
             (f2cl-lib:fref d-%data%
                           (ip1)
                           ((1 *))
                           d-%offset%))
            tau)))))))
  (cond
    (orgati
     (setf ii i)
     (setf sigma
       (+ (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) eta))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
       ((> j n) nil)
       (tagbody
        (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
              (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                 eta))
        (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
              (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                 eta))))))

```

```
(t
  (setf ii (f2cl-lib:int-add i 1))
  (setf sigma
    (+ (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%) eta))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (+ (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
          eta))
      (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
        (- (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data% (ip1) ((1 *)) d-%offset%)
          eta))))))
  (setf iim1 (f2cl-lib:int-sub ii 1))
  (setf iip1 (f2cl-lib:int-add ii 1))
  (setf dps1 zero)
  (setf psi zero)
  (setf erretm zero)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j iim1) nil)
    (tagbody
      (setf temp
        (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
          (*
            (f2cl-lib:fref work-%data%
              (j)
              ((1 *))
              work-%offset%)
            (f2cl-lib:fref delta-%data%
              (j)
              ((1 *))
              delta-%offset%))))))
      (setf psi
        (+ psi
          (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
            temp)))
      (setf dps1 (+ dps1 (* temp temp)))
      (setf erretm (+ erretm psi))))
  (setf erretm (abs erretm))
  (setf dphi zero)
  (setf phi zero)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    ((> j iip1) nil)
    (tagbody
      (setf temp
        (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
          (*
            (f2cl-lib:fref work-%data%
```

```

                                (j)
                                ((1 *))
                                work-%offset%)
(f2cl-lib:fref delta-%data%
                                (j)
                                ((1 *))
                                delta-%offset%)))
(setf phi
  (+ phi
    (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
      temp)))
(setf dphi (+ dphi (* temp temp)))
(setf erretm (+ erretm phi)))
(setf w (+ rhoinv phi psi))
(setf swtch3 nil)
(cond
  (orgati
    (if (< w zero) (setf swtch3 t)))
  (t
    (if (> w zero) (setf swtch3 t))))
(if (or (= ii 1) (= ii n)) (setf swtch3 nil))
(setf temp
  (/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
    (* (f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)
      (f2cl-lib:fref delta-%data%
        (ii)
        ((1 *))
        delta-%offset%))))))
(setf dw (+ dpsd dphi (* temp temp)))
(setf temp (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%) temp))
(setf w (+ w temp))
(setf erretm
  (+ (* eight (- phi psi))
    erretm
    (* two rhoinv)
    (* three (abs temp))
    (* (abs tau) dw)))
(cond
  ((<= (abs w) (* eps erretm))
    (go end_label)))
(cond
  ((<= w zero)
    (setf sg2lb (max sg2lb tau)))
  (t
    (setf sg2ub (min sg2ub tau))))
(setf niter (f2cl-lib:int-add niter 1))
(cond
  ((not swtch3)
    (setf dtipsq
      (*

```

```

(f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
(f2cl-lib:fref delta-%data%
  (ip1)
  ((1 *))
  delta-%offset%)))
(setf dtisq
  (* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
     (f2cl-lib:fref delta-%data%
       (i)
       ((1 *))
       delta-%offset%))))
(cond
  (orgati
   (setf c
     (+ (- w (* dtipsq dw))
        (* delsq
          (expt
            (/
              (f2cl-lib:fref z-%data%
                (i)
                ((1 *))
                z-%offset%)
              dtisq)
            2))))))
  (t
   (setf c
     (- w
        (* dtisq dw)
        (* delsq
          (expt
            (/
              (f2cl-lib:fref z-%data%
                (ip1)
                ((1 *))
                z-%offset%)
              dtipsq)
            2))))))
  (setf a (+ (* (+ dtipsq dtisq) w) (* (- dtipsq) dtisq dw)))
  (setf b (* dtipsq dtisq w))
  (cond
    ((= c zero)
     (cond
       ((= a zero)
        (cond
          (orgati
           (setf a
             (+
              (*
               (f2cl-lib:fref z-%data%
                 (i)

```

```

                                ((1 *))
                                z-%offset%)
(f2cl-lib:fref z-%data%
                                (i)
                                ((1 *))
                                z-%offset%))
(* dtipsq dtipsq (+ dpsl dphi))))))
(t
  (setf a
    (+
      (*
        (f2cl-lib:fref z-%data%
          (ip1)
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          (ip1)
          ((1 *))
          z-%offset%))
      (* dtisq dtisq (+ dpsl dphi))))))
    (setf eta (/ b a)))
  (<= a zero)
  (setf eta
    (/
      (- a
        (f2cl-lib:fsqrt
          (abs (+ (* a a) (* (- four) b c))))))
      (* two c))))
  (t
    (setf eta
      (/ (* two b)
        (+ a
          (f2cl-lib:fsqrt
            (abs (+ (* a a) (* (- four) b c))))))))))
  (t
    (setf dtiim
      (*
        (f2cl-lib:fref work-%data% (iim1) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (iim1)
          ((1 *))
          delta-%offset%)))
    (setf dtiip
      (*
        (f2cl-lib:fref work-%data% (iip1) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (iip1)
          ((1 *))
          delta-%offset%)))
    (setf temp (+ rhoinv psi phi))

```

```

(cond
  (orgati
    (setf temp1
      (/ (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)
         dtiim))
    (setf temp1 (* temp1 temp1))
    (setf c
      (+ (- temp (* dtiip (+ dps1 dphi)))
         (*
           (-
             (-
               (f2cl-lib:fref d-%data%
                             (iim1)
                             ((1 *))
                             d-%offset%)
               (f2cl-lib:fref d-%data%
                             (iip1)
                             ((1 *))
                             d-%offset%)))
           (+
             (f2cl-lib:fref d-%data%
                             (iim1)
                             ((1 *))
                             d-%offset%)
             (f2cl-lib:fref d-%data%
                             (iip1)
                             ((1 *))
                             d-%offset%)))
         temp1)))
    (setf (f2cl-lib:fref zz (1) ((1 3)))
      (* (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (iim1) ((1 *)) z-%offset%)))
    (cond
      ((< dps1 temp1)
        (setf (f2cl-lib:fref zz (3) ((1 3))) (* dtiip dtiip dphi)))
      (t
        (setf (f2cl-lib:fref zz (3) ((1 3)))
          (* dtiip dtiip (+ (- dps1 temp1) dphi))))))
    (t
      (setf temp1
        (/ (f2cl-lib:fref z-%data% (iip1) ((1 *)) z-%offset%)
           dtiip))
      (setf temp1 (* temp1 temp1))
      (setf c
        (+ (- temp (* dtiim (+ dps1 dphi)))
           (*
             (-
               (-
                 (f2cl-lib:fref d-%data%
                               (iip1)

```

```

((1 *))
d-%offset%)
(f2cl-lib:fref d-%data%
(iim1)
((1 *))
d-%offset%)))
(
(f2cl-lib:fref d-%data%
(iim1)
((1 *))
d-%offset%)
(f2cl-lib:fref d-%data%
(iip1)
((1 *))
d-%offset%))
temp1)))
(cond
((< dphi temp1)
(setf (f2cl-lib:fref zz (1) ((1 3))) (* dtiim dtiim dps)))
(t
(setf (f2cl-lib:fref zz (1) ((1 3)))
(* dtiim dtiim (+ dps (- dphi temp1)))))
(setf (f2cl-lib:fref zz (3) ((1 3)))
(* (f2cl-lib:fref z-%data% (iip1) ((1 *)) z-%offset%)
(f2cl-lib:fref z-%data%
(iip1)
((1 *))
z-%offset%))))))
(setf (f2cl-lib:fref zz (2) ((1 3)))
(* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
(f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)))
(setf (f2cl-lib:fref dd (1) ((1 3))) dtiim)
(setf (f2cl-lib:fref dd (2) ((1 3)))
(*
(f2cl-lib:fref delta-%data% (ii) ((1 *)) delta-%offset%)
(f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)))
(setf (f2cl-lib:fref dd (3) ((1 3))) dtiip)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
(dlaed6 niter orgati c dd zz w eta info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
(setf eta var-6)
(setf info var-7))
(if (/= info 0) (go end_label))))
(if (>= (* w eta) zero) (setf eta (/ (- w) dw)))
(cond
(orgati
(setf temp1
(* (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
(f2cl-lib:fref delta-%data%
```



```

                                (i)
                                ((1 *))
                                delta-%offset%)))
  (setf temp (- eta temp1)))
  (t
    (setf temp1
      (*
        (f2cl-lib:fref work-%data% (ip1) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (ip1)
          ((1 *))
          delta-%offset%)))
      (setf temp (- eta temp1))))
  (cond
    ((or (> temp sg2ub) (< temp sg2lb))
      (cond
        ((< w zero)
          (setf eta (/ (- sg2ub tau) two)))
        (t
          (setf eta (/ (- sg2lb tau) two))))))
  (setf tau (+ tau eta))
  (setf eta (/ eta (+ sigma (f2cl-lib:fsqrt (+ (* sigma sigma) eta)))))
  (setf prew w)
  (setf sigma (+ sigma eta))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j n) nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (+ (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
          eta))
      (setf (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
        (-
          (f2cl-lib:fref delta-%data% (j) ((1 *)) delta-%offset%)
          eta))))
  (setf dpsl zero)
  (setf psi zero)
  (setf erretm zero)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j iim1) nil)
    (tagbody
      (setf temp
        (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
          (*
            (f2cl-lib:fref work-%data%
              (j)
              ((1 *))
              work-%offset%)
            (f2cl-lib:fref delta-%data%
              (j)
              ((1 *))

```

```

                                delta-%offset%)))
  (setf psi
    (+ psi
      (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%
        temp)))
    (setf dpsi (+ dpsi (* temp temp)))
    (setf erretm (+ erretm psi))))
  (setf erretm (abs erretm))
  (setf dphi zero)
  (setf phi zero)
  (f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j iip1) nil)
  (tagbody
    (setf temp
      (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%
        (*
          (f2cl-lib:fref work-%data%
            (j)
            ((1 *))
            work-%offset%)
          (f2cl-lib:fref delta-%data%
            (j)
            ((1 *))
            delta-%offset%))))))
    (setf phi
      (+ phi
        (* (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%
          temp)))
      (setf dphi (+ dphi (* temp temp)))
      (setf erretm (+ erretm phi))))
  (setf temp
    (/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref work-%data% (ii) ((1 *)) work-%offset%)
        (f2cl-lib:fref delta-%data%
          (ii)
          ((1 *))
          delta-%offset%))))))
  (setf dw (+ dpsi dphi (* temp temp)))
  (setf temp (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%) temp))
  (setf w (+ rhoinv phi psi temp))
  (setf erretm
    (+ (* eight (- phi psi))
      erretm
      (* two rhoinv)
      (* three (abs temp))
      (* (abs tau) dw)))
  (cond
    ((<= w zero)
      (setf sg2lb (max sg2lb tau)))
    (t

```

```

      (setf sg2ub (min sg2ub tau))))
    (setf swtch nil)
    (cond
      (orgati
        (if (> (- w) (/ (abs prew) ten)) (setf swtch t)))
      (t
        (if (> w (/ (abs prew) ten)) (setf swtch t))))
    (setf iter (f2cl-lib:int-add niter 1))
    (f2cl-lib:fdo (niter iter (f2cl-lib:int-add niter 1))
      ((> niter maxit) nil)
      (tagbody
        (cond
          ((<= (abs w) (* eps erretm))
            (go end_label)))
        (cond
          ((not swtch3)
            (setf dtipsq
              (*
                (f2cl-lib:fref work-%data%
                               (ip1)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (ip1)
                               ((1 *))
                               delta-%offset%)))
            (setf dtisq
              (*
                (f2cl-lib:fref work-%data%
                               (i)
                               ((1 *))
                               work-%offset%)
                (f2cl-lib:fref delta-%data%
                               (i)
                               ((1 *))
                               delta-%offset%))))
          (cond
            ((not swtch)
              (cond
                (orgati
                  (setf c
                    (+ (- w (* dtipsq dw))
                      (* delsq
                        (expt
                          (/
                            (f2cl-lib:fref z-%data%
                                              (i)
                                              ((1 *))
                                              z-%offset%)
                            dtisq)

```

```

2))))))
(t
  (setf c
    (- w
      (* dtisq dw)
      (* delsq
        (expt
          (/
            (f2cl-lib:fref z-%data%
                          (ip1)
                          ((1 *))
                          z-%offset%)
            dtipsq)
          2))))))
(t
  (setf temp
    (/
      (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
      (*
        (f2cl-lib:fref work-%data%
                      (ii)
                      ((1 *))
                      work-%offset%)
        (f2cl-lib:fref delta-%data%
                      (ii)
                      ((1 *))
                      delta-%offset%))))
  (cond
    (orgati
      (setf dpsl (+ dpsl (* temp temp))))
    (t
      (setf dphi (+ dphi (* temp temp))))
    (setf c (- w (* dtisq dpsl) (* dtipsq dphi))))
  (setf a (+ (* (+ dtipsq dtisq) w) (* (- dtipsq) dtisq dw)))
  (setf b (* dtipsq dtisq w))
  (cond
    ((= c zero)
      (cond
        ((= a zero)
          (cond
            ((not swtch)
              (cond
                (orgati
                  (setf a
                    (+
                      (*
                        (f2cl-lib:fref z-%data%
                                      (i)
                                      ((1 *))
                                      z-%offset%)

```

```

                                (f2cl-lib:fref z-%data%
                                (i)
                                ((1 *))
                                z-%offset%))
                                (* dtipsq dtipsq (+ dpsi dphi))))))
(t
  (setf a
    (+
      (*
        (f2cl-lib:fref z-%data%
          (ip1)
          ((1 *))
          z-%offset%))
        (f2cl-lib:fref z-%data%
          (ip1)
          ((1 *))
          z-%offset%))
      (* dtisq dtisq (+ dpsi dphi))))))
(t
  (setf a
    (+ (* dtisq dtisq dpsi)
      (* dtipsq dtipsq dphi))))))
(setf eta (/ b a)))
(<= a zero)
(setf eta
  (/
    (- a
      (f2cl-lib:fsqrt
        (abs (+ (* a a) (* (- four) b c))))))
    (* two c)))
(t
  (setf eta
    (/ (* two b)
      (+ a
        (f2cl-lib:fsqrt
          (abs (+ (* a a) (* (- four) b c))))))))))
(t
  (setf dtiim
    (*
      (f2cl-lib:fref work-%data%
        (iim1)
        ((1 *))
        work-%offset%))
      (f2cl-lib:fref delta-%data%
        (iim1)
        ((1 *))
        delta-%offset%)))
  (setf dtiip
    (*
      (f2cl-lib:fref work-%data%

```

```

                                (iip1)
                                ((1 *))
                                work-%offset%)
(f2cl-lib:fref delta-%data%
                                (iip1)
                                ((1 *))
                                delta-%offset%)))
(setf temp (+ rhoinv psi phi))
(cond
  (swtch
    (setf c (- temp (* dtiim dps) (* dtiip dphi)))
    (setf (f2cl-lib:fref zz (1) ((1 3))) (* dtiim dtiim dps))
    (setf (f2cl-lib:fref zz (3) ((1 3))) (* dtiip dtiip dphi)))
  (t
    (cond
      (orgati
        (setf temp1
          (/
            (f2cl-lib:fref z-%data%
                          (iim1)
                          ((1 *))
                          z-%offset%)
            dtiim))
        (setf temp1 (* temp1 temp1))
        (setf temp2
          (*
            (-
              (f2cl-lib:fref d-%data%
                            (iim1)
                            ((1 *))
                            d-%offset%)
              (f2cl-lib:fref d-%data%
                            (iip1)
                            ((1 *))
                            d-%offset%))
            (+
              (f2cl-lib:fref d-%data%
                            (iim1)
                            ((1 *))
                            d-%offset%)
              (f2cl-lib:fref d-%data%
                            (iip1)
                            ((1 *))
                            d-%offset%))
            temp1))
        (setf c (- temp (* dtiip (+ dps dphi)) temp2))
        (setf (f2cl-lib:fref zz (1) ((1 3)))
          (*
            (f2cl-lib:fref z-%data%
                          (iim1)

```

```

                                ((1 *))
                                z-%offset%)
(f2cl-lib:fref z-%data%
                                (iim1)
                                ((1 *))
                                z-%offset%)))
(cond
  ((< dpsi temp1)
    (setf (f2cl-lib:fref zz (3) ((1 3)))
          (* dtiip dtiip dphi)))
  (t
    (setf (f2cl-lib:fref zz (3) ((1 3)))
          (* dtiip dtiip (+ (- dpsi temp1) dphi))))))
(t
  (setf temp1
    (/
      (f2cl-lib:fref z-%data%
                      (iip1)
                      ((1 *))
                      z-%offset%)
      dtiip))
  (setf temp1 (* temp1 temp1))
  (setf temp2
    (*
      (-
        (f2cl-lib:fref d-%data%
                        (iip1)
                        ((1 *))
                        d-%offset%)
        (f2cl-lib:fref d-%data%
                        (iim1)
                        ((1 *))
                        d-%offset%))
      (+
        (f2cl-lib:fref d-%data%
                        (iim1)
                        ((1 *))
                        d-%offset%)
        (f2cl-lib:fref d-%data%
                        (iip1)
                        ((1 *))
                        d-%offset%))
      temp1))
  (setf c (- temp (* dtiim (+ dpsi dphi)) temp2))
  (cond
    ((< dphi temp1)
      (setf (f2cl-lib:fref zz (1) ((1 3)))
            (* dtiim dtiim dpsi)))
    (t
      (setf (f2cl-lib:fref zz (1) ((1 3)))
            (* dtiim dtiim dpsi))))

```

```

(* dtiim dtiim (+ dpsl (- dphi temp1))))))
(setf (f2cl-lib:fref zz (3) ((1 3)))
  (*
    (f2cl-lib:fref z-%data%
      (iip1)
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      (iip1)
      ((1 *))
      z-%offset%))))))
(setf (f2cl-lib:fref dd (1) ((1 3))) dtiim)
(setf (f2cl-lib:fref dd (2) ((1 3)))
  (*
    (f2cl-lib:fref delta-%data%
      (ii)
      ((1 *))
      delta-%offset%)
    (f2cl-lib:fref work-%data%
      (ii)
      ((1 *))
      work-%offset%))))
(setf (f2cl-lib:fref dd (3) ((1 3))) dtiip)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dlaed6 niter orgati c dd zz w eta info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5))
  (setf eta var-6)
  (setf info var-7))
  (if (/= info 0) (go end_label))))
(if (>= (* w eta) zero) (setf eta (/ (- w) dw)))
(cond
  (orgati
    (setf temp1
      (*
        (f2cl-lib:fref work-%data%
          (i)
          ((1 *))
          work-%offset%)
        (f2cl-lib:fref delta-%data%
          (i)
          ((1 *))
          delta-%offset%))))
    (setf temp (- eta temp1)))
  (t
    (setf temp1
      (*
        (f2cl-lib:fref work-%data%
          (ip1)
          ((1 *))

```



```

                                work-%offset%)
      (f2cl-lib:fref delta-%data%
        (ip1)
        ((1 *))
        delta-%offset%)))
    (setf temp (- eta temp1))))
  (cond
    ((or (> temp sg2ub) (< temp sg2lb))
     (cond
       ((< w zero)
        (setf eta (/ (- sg2ub tau) two)))
       (t
        (setf eta (/ (- sg2lb tau) two))))))
    (setf tau (+ tau eta))
    (setf eta
      (/ eta
        (+ sigma (f2cl-lib:fsqrt (+ (* sigma sigma) eta))))))
    (setf sigma (+ sigma eta))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
          (+
            (f2cl-lib:fref work-%data%
              (j)
              ((1 *))
              work-%offset%)
            eta))
          (setf (f2cl-lib:fref delta-%data%
            (j)
            ((1 *))
            delta-%offset%)
              (-
                (f2cl-lib:fref delta-%data%
                  (j)
                  ((1 *))
                  delta-%offset%)
                eta))))))
    (setf prew w)
    (setf dpsi zero)
    (setf psi zero)
    (setf erretm zero)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j iim1) nil)
      (tagbody
        (setf temp
          (/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
            *
            (f2cl-lib:fref work-%data%
              (j)
              ((1 *))
              work-%offset%)
            eta))))))

```

```

((1 *))
work-%offset%)
(f2cl-lib:fref delta-%data%
(j)
((1 *))
delta-%offset%))))

(setf psi
(+ psi
(*
(f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
temp)))
(setf dpsi (+ dpsi (* temp temp)))
(setf erretm (+ erretm psi))))
(setf erretm (abs erretm))
(setf dphi zero)
(setf phi zero)
(f2cl-lib:fdof (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
(> j iip1) nil)
(tagbody
(setf temp
(/ (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
(*
(f2cl-lib:fref work-%data%
(j)
((1 *))
work-%offset%)
(f2cl-lib:fref delta-%data%
(j)
((1 *))
delta-%offset%))))))

(setf phi
(+ phi
(*
(f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
temp)))
(setf dphi (+ dphi (* temp temp)))
(setf erretm (+ erretm phi))))
(setf temp
(/ (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
(*
(f2cl-lib:fref work-%data%
(ii)
((1 *))
work-%offset%)
(f2cl-lib:fref delta-%data%
(ii)
((1 *))
delta-%offset%))))))
(setf dw (+ dpsi dphi (* temp temp)))
(setf temp

```

```

                (* (f2cl-lib:fref z-%data% (ii) ((1 *)) z-%offset%)
                   temp))
      (setf w (+ rhoinv phi psi temp))
      (setf erretm
        (+ (* eight (- phi psi))
           erretm
           (* two rhoinv)
           (* three (abs temp))
           (* (abs tau) dw)))
      (if (and (> (* w prew) zero) (> (abs w) (/ (abs prew) ten)))
          (setf swtch (not swtch)))
      (cond
        ((<= w zero)
         (setf sg2lb (max sg2lb tau)))
        (t
         (setf sg2ub (min sg2ub tau))))))
      (setf info 1)))
end_label
(return (values nil nil nil nil nil nil sigma nil info))))

```

dlasd5 LAPACK

— dlasd5.input —

```

)set break resume
)sys rm -f dlasd5.output
)spool dlasd5.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd5.help —

```

=====
dlasd5 examples
=====
=====

```

Man Page Details

=====

NAME

DLASD5 - compute the square root of the I-th eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix $\text{diag}(D) * \text{diag}(D) + \text{RHO}$. The diagonal entries in the array D are assumed to satisfy $0 \leq D(i) < D(j)$ for $i < j$.

SYNOPSIS

SUBROUTINE DLASD5(I, D, Z, DELTA, RHO, DSIGMA, WORK)

INTEGER I

DOUBLE PRECISION DSIGMA, RHO

DOUBLE PRECISION D(2), DELTA(2), WORK(2), Z(2)

Purpose

=====

This subroutine computes the square root of the I-th eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix

$$\text{diag}(D) * \text{diag}(D) + \text{RHO} * Z * \text{transpose}(Z).$$

The diagonal entries in the array D are assumed to satisfy

$$0 \leq D(i) < D(j) \text{ for } i < j.$$

We also assume $\text{RHO} > 0$ and that the Euclidean norm of the vector Z is one.

Arguments

=====

I (input) INTEGER

The index of the eigenvalue to be computed. $I = 1$ or $I = 2$.

D (input) DOUBLE PRECISION array, dimension (2)

The original eigenvalues. We assume $0 \leq D(1) < D(2)$.

Z (input) DOUBLE PRECISION array, dimension (2)

The components of the updating vector.

DELTA (output) DOUBLE PRECISION array, dimension (2)

Contains $(D(j) - \text{lambda}_I)$ in its j-th component.

The vector DELTA contains the information necessary to construct the eigenvectors.

RHO (input) DOUBLE PRECISION
The scalar in the symmetric updating formula.

DSIGMA (output) DOUBLE PRECISION
The computed λ_I , the I-th updated eigenvalue.

WORK (workspace) DOUBLE PRECISION array, dimension (2)
WORK contains $D(j) + \sigma_I$ in its j-th component.

Further Details
=====

Based on contributions by
Ren-Cang Li, Computer Science Division, University of California
at Berkeley, USA

— dlasd5.f —

```

SUBROUTINE DLASD5( I, D, Z, DELTA, RHO, DSIGMA, WORK )
*
* -- LAPACK auxiliary routine (version 3.0) --
* Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
* Courant Institute, NAG Ltd., and Rice University
* June 30, 1999
*
* .. Scalar Arguments ..
INTEGER          I
DOUBLE PRECISION DSIGMA, RHO
*
* ..
* .. Array Arguments ..
DOUBLE PRECISION D( 2 ), DELTA( 2 ), WORK( 2 ), Z( 2 )
*
* ..
*
* =====
*
* .. Parameters ..
DOUBLE PRECISION ZERO, ONE, TWO, THREE, FOUR
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0,
$                 THREE = 3.0D+0, FOUR = 4.0D+0 )
*
* ..
* .. Local Scalars ..
DOUBLE PRECISION B, C, DEL, DELSQ, TAU, W
*
* ..
* .. Intrinsic Functions ..
INTRINSIC        ABS, SQRT

```

```

*      ..
*      .. Executable Statements ..
*
      DEL = D( 2 ) - D( 1 )
      DELSQ = DEL*( D( 2 )+D( 1 ) )
      IF( I.EQ.1 ) THEN
        W = ONE + FOUR*RHO*( Z( 2 )*Z( 2 ) / ( D( 1 )+THREE*D( 2 ) )-
$      Z( 1 )*Z( 1 ) / ( THREE*D( 1 )+D( 2 ) ) ) / DEL
        IF( W.GT.ZERO ) THEN
          B = DELSQ + RHO*( Z( 1 )*Z( 1 )+Z( 2 )*Z( 2 ) )
          C = RHO*Z( 1 )*Z( 1 )*DELSQ
*
*      B > ZERO, always
*
*      The following TAU is DSIGMA * DSIGMA - D( 1 ) * D( 1 )
*
          TAU = TWO*C / ( B+SQRT( ABS( B*B-FOUR*C ) ) )
*
*      The following TAU is DSIGMA - D( 1 )
*
          TAU = TAU / ( D( 1 )+SQRT( D( 1 )*D( 1 )+TAU ) )
          DSIGMA = D( 1 ) + TAU
          DELTA( 1 ) = -TAU
          DELTA( 2 ) = DEL - TAU
          WORK( 1 ) = TWO*D( 1 ) + TAU
          WORK( 2 ) = ( D( 1 )+TAU ) + D( 2 )
*      DELTA( 1 ) = -Z( 1 ) / TAU
*      DELTA( 2 ) = Z( 2 ) / ( DEL-TAU )
        ELSE
          B = -DELSQ + RHO*( Z( 1 )*Z( 1 )+Z( 2 )*Z( 2 ) )
          C = RHO*Z( 2 )*Z( 2 )*DELSQ
*
*      The following TAU is DSIGMA * DSIGMA - D( 2 ) * D( 2 )
*
          IF( B.GT.ZERO ) THEN
            TAU = -TWO*C / ( B+SQRT( B*B+FOUR*C ) )
          ELSE
            TAU = ( B-SQRT( B*B+FOUR*C ) ) / TWO
          END IF
*
*      The following TAU is DSIGMA - D( 2 )
*
          TAU = TAU / ( D( 2 )+SQRT( ABS( D( 2 )*D( 2 )+TAU ) ) )
          DSIGMA = D( 2 ) + TAU
          DELTA( 1 ) = -( DEL+TAU )
          DELTA( 2 ) = -TAU
          WORK( 1 ) = D( 1 ) + TAU + D( 2 )
          WORK( 2 ) = TWO*D( 2 ) + TAU
*      DELTA( 1 ) = -Z( 1 ) / ( DEL+TAU )
*      DELTA( 2 ) = -Z( 2 ) / TAU

```

```

      END IF
*      TEMP = SQRT( DELTA( 1 )*DELTA( 1 )+DELTA( 2 )*DELTA( 2 ) )
*      DELTA( 1 ) = DELTA( 1 ) / TEMP
*      DELTA( 2 ) = DELTA( 2 ) / TEMP
      ELSE
*
*      Now I=2
*
      B = -DELSQ + RHO*( Z( 1 )*Z( 1 )+Z( 2 )*Z( 2 ) )
      C = RHO*Z( 2 )*Z( 2 )*DELSQ
*
*      The following TAU is DSIGMA * DSIGMA - D( 2 ) * D( 2 )
*
      IF( B.GT.ZERO ) THEN
          TAU = ( B+SQRT( B*B+FOUR*C ) ) / TWO
      ELSE
          TAU = TWO*C / ( -B+SQRT( B*B+FOUR*C ) )
      END IF
*
*      The following TAU is DSIGMA - D( 2 )
*
      TAU = TAU / ( D( 2 )+SQRT( D( 2 )*D( 2 )+TAU ) )
      DSIGMA = D( 2 ) + TAU
      DELTA( 1 ) = -( DEL+TAU )
      DELTA( 2 ) = -TAU
      WORK( 1 ) = D( 1 ) + TAU + D( 2 )
      WORK( 2 ) = TWO*D( 2 ) + TAU
*      DELTA( 1 ) = -Z( 1 ) / ( DEL+TAU )
*      DELTA( 2 ) = -Z( 2 ) / TAU
*      TEMP = SQRT( DELTA( 1 )*DELTA( 1 )+DELTA( 2 )*DELTA( 2 ) )
*      DELTA( 1 ) = DELTA( 1 ) / TEMP
*      DELTA( 2 ) = DELTA( 2 ) / TEMP
      END IF
      RETURN
*
*      End of DLASD5
*
      END

```

— LAPACK dlasd5 —

```

(let* ((zero 0.0) (one 1.0) (two 2.0) (three 3.0) (four 4.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 3.0 3.0) three)

```

```

        (type (double-float 4.0 4.0) four))
(defun dlasd5 (i d z delta rho dsigma work)
  (declare (type (double-float) dsigma rho)
    (type (simple-array double-float (*)) work delta z d)
    (type fixnum i))
  (f2cl-lib:with-multi-array-data
    ((d double-float d-%data% d-%offset%)
     (z double-float z-%data% z-%offset%)
     (delta double-float delta-%data% delta-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((b 0.0) (c 0.0) (del 0.0) (delsq 0.0) (tau 0.0) (w 0.0))
      (declare (type (double-float) b c del delsq tau w))
      (setf del
        (- (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
           (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)))
      (setf delsq
        (* del
          (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
             (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%))))
      (cond
        ((= i 1)
         (setf w
           (+ one
              (/
               (* four
                  rho
                  (+
                     (/
                      (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
                        (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%))
                      (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                        (* three
                           (f2cl-lib:fref d-%data%
                                           (2)
                                           ((1 2))
                                           d-%offset%))))
                     (/
                      (*
                        (-
                          (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
                          (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
                        (+
                          (* three
                             (f2cl-lib:fref d-%data%
                                           (1)
                                           ((1 2))
                                           d-%offset%))
                          (f2cl-lib:fref d-%data%
                                           (2)
                                           ((1 2))
                                           d-%offset%))))
                      ))
                  ))
              ))
          ))
         ))

```



```

                                d-%offset%))))))
                                del))))
(cond
  (> w zero)
  (setf b
    (+ delsq
      (* rho
        (+
          (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data%
                          (1)
                          ((1 2))
                          z-%offset%))
          (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data%
                          (2)
                          ((1 2))
                          z-%offset%)))))))
    (setf c
      (* rho
        (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
        (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
        delsq))
    (setf tau
      (/ (* two c)
        (+ b (f2cl-lib:fsqrt (abs (- (* b b) (* four c)))))))
    (setf tau
      (/ tau
        (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
          (f2cl-lib:fsqrt
            (+
              (*
                (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
                tau))))))
    (setf dsigma
      (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%) tau))
    (setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
      (- tau))
    (setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
      (- del tau))
    (setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
      (+
        (* two (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
          tau))
    (setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)
      (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
        tau
        (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))))
    (t

```

```

(setf b
  (-
    (* rho
      (+
        (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
          (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
        (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
          (f2cl-lib:fref z-%data%
            (2)
            ((1 2))
            z-%offset%))))))
    delsq))
(setf c
  (* rho
    (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
    (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
    delsq))
(cond
  ((> b zero)
    (setf tau
      (/ (* (- two) c)
        (+ b (f2cl-lib:fsqrt (+ (* b b) (* four c)))))))
  (t
    (setf tau
      (/ (- b (f2cl-lib:fsqrt (+ (* b b) (* four c))))
        two))))
(setf tau
  (/ tau
    (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
      (f2cl-lib:fsqrt
        (abs
          (+
            (*
              (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
              (f2cl-lib:fref d-%data%
                (2)
                ((1 2))
                d-%offset%))
            tau))))))
(setf dsigma
  (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%) tau))
(setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
  (- (+ del tau)))
(setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
  (- tau))
(setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
  (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
    tau
    (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)))
(setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)

```

```

      (+
      (* two (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
      tau))))))
(t
  (setf b
    (-
      (* rho
        (+
          (* (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data% (1) ((1 2)) z-%offset%))
          (* (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
            (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%))))
        delsq))
    (setf c
      (* rho
        (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
        (f2cl-lib:fref z-%data% (2) ((1 2)) z-%offset%)
        delsq))
    (cond
      ((> b zero)
        (setf tau (/ (+ b (f2cl-lib:fsqrt (+ (* b b) (* four c)))) two)))
      (t
        (setf tau
          (/ (* two c)
            (- (f2cl-lib:fsqrt (+ (* b b) (* four c))) b))))))
  (setf tau
    (/ tau
      (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
        (f2cl-lib:fsqrt
          (+
            (* (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
              (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%))
            tau))))))
  (setf dsigma (+ (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%) tau))
  (setf (f2cl-lib:fref delta-%data% (1) ((1 2)) delta-%offset%)
    (- (+ del tau)))
  (setf (f2cl-lib:fref delta-%data% (2) ((1 2)) delta-%offset%)
    (- tau))
  (setf (f2cl-lib:fref work-%data% (1) ((1 2)) work-%offset%)
    (+ (f2cl-lib:fref d-%data% (1) ((1 2)) d-%offset%)
      tau
      (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)))
  (setf (f2cl-lib:fref work-%data% (2) ((1 2)) work-%offset%)
    (+ (* two (f2cl-lib:fref d-%data% (2) ((1 2)) d-%offset%)
      tau))))
  (return (values nil nil nil nil nil dsigma nil))))))

```

dlasd6 LAPACK**— dlasd6.input —**

```

)set break resume
)sys rm -f dlasd6.output
)spool dlasd6.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd6.help —

```

=====
dlasd6 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASD6 - the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row

SYNOPSIS

```

SUBROUTINE DLASD6( ICOMPQ, NL, NR, SQRE, D, VF, VL, ALPHA, BETA, IDXQ,
                  PERM, GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM, POLES,
                  DIFL, DIFR, Z, K, C, S, WORK, IWORK, INFO )

```

```

      INTEGER      GIVPTR, ICOMPQ, INFO, K, LDGCOL, LDGNUM, NL, NR,
                  SQRE

```

```

      DOUBLE      PRECISION ALPHA, BETA, C, S

```

```

      INTEGER      GIVCOL( LDGCOL, * ), IDXQ( * ), IWORK( * ), PERM( * )

```

```

      DOUBLE      PRECISION D( * ), DIFL( * ), DIFR( * ), GIVNUM(
                  LDGNUM, * ), POLES( LDGNUM, * ), VF( * ), VL( * ),
                  WORK( * ), Z( * )

```

Purpose

=====

DLASD6 computes the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row. This routine is used only for the problem which requires all singular values and optionally singular vector matrices in factored form. B is an N-by-M matrix with $N = NL + NR + 1$ and $M = N + SQRE$. A related subroutine, DLASD1, handles the case in which all singular values and singular vectors of the bidiagonal matrix are desired.

DLASD6 computes the SVD as follows:

$$B = U(in) * \begin{pmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{pmatrix} * VT(in)$$

$$= U(out) * \begin{pmatrix} D(out) & 0 \end{pmatrix} * VT(out)$$

where $Z' = (Z1' \ a \ Z2' \ b) = u' \ VT'$, and u is a vector of dimension M with ALPHA and BETA in the NL+1 and NL+2 th entries and zeros elsewhere; and the entry b is empty if SQRE = 0.

The singular values of B can be computed using D1, D2, the first components of all the right singular vectors of the lower block, and the last components of all the right singular vectors of the upper block. These components are stored and updated in VF and VL, respectively, in DLASD6. Hence U and VT are not explicitly referenced.

The singular values are stored in D. The algorithm consists of two stages:

The first stage consists of deflating the size of the problem when there are multiple singular values or if there is a zero in the Z vector. For each such occurrence the dimension of the secular equation problem is reduced by one. This stage is performed by the routine DLASD7.

The second stage consists of calculating the updated singular values. This is done by finding the roots of the secular equation via the routine DLASD4 (as called by DLASD8). This routine also updates VF and VL and computes the distances between the updated singular values and the old singular values.

DLASD6 is called from DLASDA.

Arguments

=====

ICOMPQ (input) INTEGER
 Specifies whether singular vectors are to be computed in factored form:
 = 0: Compute singular values only.
 = 1: Compute singular vectors in factored form as well.

NL (input) INTEGER
 The row dimension of the upper block. $NL \geq 1$.

NR (input) INTEGER
 The row dimension of the lower block. $NR \geq 1$.

SQRE (input) INTEGER
 = 0: the lower block is an NR -by- NR square matrix.
 = 1: the lower block is an NR -by- $(NR+1)$ rectangular matrix.

The bidiagonal matrix has row dimension $N = NL + NR + 1$, and column dimension $M = N + SQRE$.

D (input/output) DOUBLE PRECISION array, dimension ($NL+NR+1$).
 On entry $D(1:NL,1:NL)$ contains the singular values of the upper block, and $D(NL+2:N)$ contains the singular values of the lower block. On exit $D(1:N)$ contains the singular values of the modified matrix.

VF (input/output) DOUBLE PRECISION array, dimension (M)
 On entry, $VF(1:NL+1)$ contains the first components of all right singular vectors of the upper block; and $VF(NL+2:M)$ contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VL (input/output) DOUBLE PRECISION array, dimension (M)
 On entry, $VL(1:NL+1)$ contains the last components of all right singular vectors of the upper block; and $VL(NL+2:M)$ contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

ALPHA (input) DOUBLE PRECISION
 Contains the diagonal element associated with the added row.

BETA (input) DOUBLE PRECISION
 Contains the off-diagonal element associated with the added row.

IDXQ (output) INTEGER array, dimension (N)
 This contains the permutation which will reintegrate the subproblem just solved back into sorted order, i.e.
 $D(\text{IDXQ}(I = 1, N))$ will be in ascending order.

PERM (output) INTEGER array, dimension (N)
The permutations (from deflation and sorting) to be applied to each block. Not referenced if ICOMPQ = 0.

GIVPTR (output) INTEGER
The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL (output) INTEGER array, dimension (LDGCOL, 2)
Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL (input) INTEGER
leading dimension of GIVCOL, must be at least N.

GIVNUM (output) DOUBLE PRECISION array, dimension (LDGNUM, 2)
Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM (input) INTEGER
The leading dimension of GIVNUM and POLES, must be at least N.

POLES (output) DOUBLE PRECISION array, dimension (LDGNUM, 2)
On exit, POLES(1,*) is an array containing the new singular values obtained from solving the secular equation, and POLES(2,*) is an array containing the poles in the secular equation. Not referenced if ICOMPQ = 0.

DIFL (output) DOUBLE PRECISION array, dimension (N)
On exit, DIFL(I) is the distance between I-th updated (undeflated) singular value and the I-th (undeflated) old singular value.

DIFR (output) DOUBLE PRECISION array,
dimension (LDGNUM, 2) if ICOMPQ = 1 and
dimension (N) if ICOMPQ = 0.
On exit, DIFR(I, 1) is the distance between I-th updated (undeflated) singular value and the I+1-th (undeflated) old singular value.

If ICOMPQ = 1, DIFR(1:K,2) is an array containing the normalizing factors for the right singular vector matrix.

See DLASD8 for details on DIFL and DIFR.

Z (output) DOUBLE PRECISION array, dimension (M)
The first elements of this array contain the components of the deflation-adjusted updating row vector.

K (output) INTEGER
 Contains the dimension of the non-deflated matrix,
 This is the order of the related secular equation. $1 \leq K \leq N$.

C (output) DOUBLE PRECISION
 C contains garbage if SQRE = 0 and the C-value of a Givens
 rotation related to the right null space if SQRE = 1.

S (output) DOUBLE PRECISION
 S contains garbage if SQRE = 0 and the S-value of a Givens
 rotation related to the right null space if SQRE = 1.

WORK (workspace) DOUBLE PRECISION array, dimension (4 * M)

IWORK (workspace) INTEGER array, dimension (3 * N)

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.
 > 0: if INFO = 1, an singular value did not converge

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of
 California at Berkeley, USA

— dlasd6.f —

```

SUBROUTINE DLASD6( ICOMPQ, NL, NR, SQRE, D, VF, VL, ALPHA, BETA,
$                 IDXQ, PERM, GIVPTR, GIVCOL, LDGCOL, GIVNUM,
$                 LDGNUM, POLES, DIFL, DIFR, Z, K, C, S, WORK,
$                 IWORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
INTEGER          GIVPTR, ICOMPQ, INFO, K, LDGCOL, LDGNUM, NL,
$              NR, SQRE
DOUBLE PRECISION ALPHA, BETA, C, S
*
*  .. Array Arguments ..

```



```

      INTEGER          GIVCOL( LDGCOL, * ), IDXQ( * ), IWORK( * ),
$      PERM( * )
      DOUBLE PRECISION D( * ), DIFL( * ), DIFR( * ),
$      GIVNUM( LDGNUM, * ), POLES( LDGNUM, * ),
$      VF( * ), VL( * ), WORK( * ), Z( * )
*      ..
*
* =====
*
*      .. Parameters ..
      DOUBLE PRECISION  ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*      ..
*      .. Local Scalars ..
      INTEGER            I, IDX, IDXC, IDXP, ISIGMA, IVFW, IVLW, IW, M,
$      N, N1, N2
      DOUBLE PRECISION  ORGNRM
*      ..
*      .. External Subroutines ..
      EXTERNAL           DCOPY, DLAMRG, DLASCL, DLASD7, DLASD8, XERBLA
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters.
*
      INFO = 0
      N = NL + NR + 1
      M = N + SQRE
*
      IF( ( ICOMPQ.LT.0 ) .OR. ( ICOMPQ.GT.1 ) ) THEN
         INFO = -1
      ELSE IF( NL.LT.1 ) THEN
         INFO = -2
      ELSE IF( NR.LT.1 ) THEN
         INFO = -3
      ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN
         INFO = -4
      ELSE IF( LDGCOL.LT.N ) THEN
         INFO = -14
      ELSE IF( LDGNUM.LT.N ) THEN
         INFO = -16
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DLASD6', -INFO )
         RETURN
      END IF
*

```

```

*      The following values are for bookkeeping purposes only.  They are
*      integer pointers which indicate the portion of the workspace
*      used by a particular array in DLASD7 and DLASD8.
*
      ISIGMA = 1
      IW = ISIGMA + N
      IVFW = IW + M
      IVLW = IVFW + M
*
      IDX = 1
      IDXC = IDX + N
      IDXP = IDXC + N
*
*      Scale.
*
      ORGNRM = MAX( ABS( ALPHA ), ABS( BETA ) )
      D( NL+1 ) = ZERO
      DO 10 I = 1, N
        IF( ABS( D( I ) ) .GT. ORGNRM ) THEN
          ORGNRM = ABS( D( I ) )
        END IF
10 CONTINUE
      CALL DLASCL( 'G', 0, 0, ORGNRM, ONE, N, 1, D, N, INFO )
      ALPHA = ALPHA / ORGNRM
      BETA = BETA / ORGNRM
*
*      Sort and Deflate singular values.
*
      CALL DLASD7( ICOMPQ, NL, NR, SQRE, K, D, Z, WORK( IW ), VF,
$                WORK( IVFW ), VL, WORK( IVLW ), ALPHA, BETA,
$                WORK( ISIGMA ), IWORK( IDX ), IWORK( IDXP ), IDXQ,
$                PERM, GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM, C, S,
$                INFO )
*
*      Solve Secular Equation, compute DIFL, DIFR, and update VF, VL.
*
      CALL DLASD8( ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDGNUM,
$                WORK( ISIGMA ), WORK( IW ), INFO )
*
*      Save the poles if ICOMPQ = 1.
*
      IF( ICOMPQ.EQ.1 ) THEN
        CALL DCOPY( K, D, 1, POLES( 1, 1 ), 1 )
        CALL DCOPY( K, WORK( ISIGMA ), 1, POLES( 1, 2 ), 1 )
      END IF
*
*      Unscale.
*
      CALL DLASCL( 'G', 0, 0, ONE, ORGNRM, N, 1, D, N, INFO )
*

```

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
           (type (double-float 0.0 0.0) zero))
  (defun dlasd6
    (icompq nl nr sqre d vf vl alpha beta idxq perm givptr givcol ldgcol
     givnum ldgnum poles difl difr z k c s work iwork info)
    (declare (type (simple-array fixnum (*)) iwork givcol perm idxq)
             (type (double-float) s c beta alpha)
             (type (simple-array double-float (*)) work z difr
                  difl poles givnum vl vf d)
             (type fixnum info k ldgnum ldgcol givptr sqre nr nl
                  icompq)))
  (f2cl-lib:with-multi-array-data
    ((d double-float d-%data% d-%offset%)
     (vf double-float vf-%data% vf-%offset%)
     (vl double-float vl-%data% vl-%offset%)
     (givnum double-float givnum-%data% givnum-%offset%)
     (poles double-float poles-%data% poles-%offset%)
     (difl double-float difl-%data% difl-%offset%)
     (difr double-float difr-%data% difr-%offset%)
     (z double-float z-%data% z-%offset%)
     (work double-float work-%data% work-%offset%)
     (idxq fixnum idxq-%data% idxq-%offset%)
     (perm fixnum perm-%data% perm-%offset%)
     (givcol fixnum givcol-%data% givcol-%offset%)
     (iwork fixnum iwork-%data% iwork-%offset%))
    (prog ((orgnrm 0.0) (i 0) (idx 0) (idxc 0) (idxp 0) (isigma 0) (ivfw 0)
          (ivlw 0) (iw 0) (m 0) (n 0) (n1 0) (n2 0))
      (declare (type (double-float) orgnrm)
               (type fixnum i idx idxc idxp isigma ivfw ivlw iw
                    m n n1 n2))
      (setf info 0)

```

```

(setf n (f2cl-lib:int-add nl nr 1))
(setf m (f2cl-lib:int-add n sqre))
(cond
  ((or (< icompq 0) (> icompq 1))
    (setf info -1))
  (< nl 1)
    (setf info -2))
  (< nr 1)
    (setf info -3))
  ((or (< sqre 0) (> sqre 1))
    (setf info -4))
  (< ldgcol n)
    (setf info -14))
  (< ldgnum n)
    (setf info -16)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASD6" (f2cl-lib:int-sub info))
  (go end_label)))
(setf isigma 1)
(setf iw (f2cl-lib:int-add isigma n))
(setf ivfw (f2cl-lib:int-add iw m))
(setf ivlw (f2cl-lib:int-add ivfw m))
(setf idx 1)
(setf idxc (f2cl-lib:int-add idx n))
(setf idxp (f2cl-lib:int-add idxc n))
(setf orgnrm (max (abs alpha) (abs beta)))
(setf (f2cl-lib:fref d-%data%
                    ((f2cl-lib:int-add nl 1))
                    ((1 *))
                    d-%offset%)
      zero)
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
  (tagbody
    (cond
      (> (abs (f2cl-lib:fref d (i) ((1 *)))) orgnrm)
      (setf orgnrm
        (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 orgnrm one n 1 d n info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf info var-9))
(setf alpha (/ alpha orgnrm))
(setf beta (/ beta orgnrm))
(multiple-value-bind

```

```
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
  var-10 var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18
  var-19 var-20 var-21 var-22 var-23 var-24 var-25 var-26)
(dlasd7 icompq nl nr sqre k d z
  (f2cl-lib:array-slice work double-float (iw) ((1 *))) vf
  (f2cl-lib:array-slice work double-float (ivfw) ((1 *))) vl
  (f2cl-lib:array-slice work double-float (ivlw) ((1 *))) alpha beta
  (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
  (f2cl-lib:array-slice iwork fixnum (idx) ((1 *)))
  (f2cl-lib:array-slice iwork fixnum (idxp) ((1 *))) idxq
  perm givptr givcol ldgcol givnum ldgnum c s info)
(declare (ignore var-0 var-1 var-2 var-3 var-5 var-6 var-7 var-8
  var-9 var-10 var-11 var-12 var-13 var-14 var-15
  var-16 var-17 var-18 var-20 var-21 var-22 var-23))
(setf k var-4)
(setf givptr var-19)
(setf c var-24)
(setf s var-25)
(setf info var-26))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11)
  (dlasd8 icompq k d z vf vl difl difr ldgnum
    (f2cl-lib:array-slice work double-float (isigma) ((1 *)))
    (f2cl-lib:array-slice work double-float (iw) ((1 *))) info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10))
  (setf info var-11))
(cond
  ((= icompq 1)
   (dcopy k d 1
     (f2cl-lib:array-slice poles double-float (1 1) ((1 ldgnum) (1 *)))
     1)
   (dcopy k (f2cl-lib:array-slice work double-float (isigma) ((1 *))) 1
     (f2cl-lib:array-slice poles double-float (1 2) ((1 ldgnum) (1 *)))
     1)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 one orgnrm n 1 d n info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf info var-9))
(setf n1 k)
(setf n2 (f2cl-lib:int-sub n k))
(dlamrg n1 n2 d 1 -1 idxq)
end_label
(return
  (values nil
    nil
    nil
```

```

nil
nil
nil
nil
alpha
beta
nil
nil
givptr
nil
nil
nil
nil
nil
nil
nil
nil
nil
k
c
s
nil
nil
info))))))

```

dlasd7 LAPACK

— dlasd7.input —

```

)set break resume
)sys rm -f dlasd7.output
)spool dlasd7.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd7.help —

```
=====
```

dlasd7 examples

=====

=====

Man Page Details

=====

NAME

DLASD7 - the two sets of singular values together into a single sorted set

SYNOPSIS

```
SUBROUTINE DLASD7( ICOMPQ, NL, NR, SQRE, K, D, Z, ZW, VF, VFW, VL, VLW,
                   ALPHA, BETA, DSIGMA, IDX, IDXP, IDXQ, PERM, GIVPTR,
                   GIVCOL, LDGCOL, GIVNUM, LDGNUM, C, S, INFO )

    INTEGER          GIVPTR, ICOMPQ, INFO, K, LDGCOL, LDGNUM, NL, NR,
                     SQRE

    DOUBLE           PRECISION ALPHA, BETA, C, S

    INTEGER          GIVCOL( LDGCOL, * ), IDX( * ), IDXP( * ), IDXQ( * ),
                     PERM( * )

    DOUBLE           PRECISION D( * ), DSIGMA( * ), GIVNUM( LDGNUM, * ),
                     VF( * ), VFW( * ), VL( * ), VLW( * ), Z( * ), ZW( * )
)
```

Purpose

=====

DLASD7 merges the two sets of singular values together into a single sorted set. Then it tries to deflate the size of the problem. There are two ways in which deflation can occur: when two or more singular values are close together or if there is a tiny entry in the Z vector. For each such occurrence the order of the related secular equation problem is reduced by one.

DLASD7 is called from DLASD6.

Arguments

=====

ICOMPQ (input) INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows:

= 0: Compute singular values only.

= 1: Compute singular vectors of upper
bidiagonal matrix in compact form.

NL (input) INTEGER
The row dimension of the upper block. $NL \geq 1$.

NR (input) INTEGER
The row dimension of the lower block. $NR \geq 1$.

SQRE (input) INTEGER
= 0: the lower block is an NR -by- NR square matrix.
= 1: the lower block is an NR -by- $(NR+1)$ rectangular matrix.

The bidiagonal matrix has
 $N = NL + NR + 1$ rows and
 $M = N + SQRE \geq N$ columns.

K (output) INTEGER
Contains the dimension of the non-deflated matrix, this is the order of the related secular equation. $1 \leq K \leq N$.

D (input/output) DOUBLE PRECISION array, dimension (N)
On entry D contains the singular values of the two submatrices to be combined. On exit D contains the trailing $(N-K)$ updated singular values (those which were deflated) sorted into increasing order.

Z (output) DOUBLE PRECISION array, dimension (M)
On exit Z contains the updating row vector in the secular equation.

ZW (workspace) DOUBLE PRECISION array, dimension (M)
Workspace for Z.

VF (input/output) DOUBLE PRECISION array, dimension (M)
On entry, $VF(1:NL+1)$ contains the first components of all right singular vectors of the upper block; and $VF(NL+2:M)$ contains the first components of all right singular vectors of the lower block. On exit, VF contains the first components of all right singular vectors of the bidiagonal matrix.

VFW (workspace) DOUBLE PRECISION array, dimension (M)
Workspace for VF.

VL (input/output) DOUBLE PRECISION array, dimension (M)
On entry, $VL(1:NL+1)$ contains the last components of all right singular vectors of the upper block; and $VL(NL+2:M)$ contains the last components of all right singular vectors of the lower block. On exit, VL contains the last components of all right singular vectors of the bidiagonal matrix.

VLW (workspace) DOUBLE PRECISION array, dimension (M)
Workspace for VL.

ALPHA (input) DOUBLE PRECISION
Contains the diagonal element associated with the added row.

BETA (input) DOUBLE PRECISION
Contains the off-diagonal element associated with the added row.

DSIGMA (output) DOUBLE PRECISION array, dimension (N)
Contains a copy of the diagonal elements (K-1 singular values and one zero) in the secular equation.

IDX (workspace) INTEGER array, dimension (N)
This will contain the permutation used to sort the contents of D into ascending order.

IDXP (workspace) INTEGER array, dimension (N)
This will contain the permutation used to place deflated values of D at the end of the array. On output IDXP(2:K) points to the nondeflated D-values and IDXP(K+1:N) points to the deflated singular values.

IDXQ (input) INTEGER array, dimension (N)
This contains the permutation which separately sorts the two sub-problems in D into ascending order. Note that entries in the first half of this permutation must first be moved one position backward; and entries in the second half must first have NL+1 added to their values.

PERM (output) INTEGER array, dimension (N)
The permutations (from deflation and sorting) to be applied to each singular block. Not referenced if ICOMPQ = 0.

GIVPTR (output) INTEGER
The number of Givens rotations which took place in this subproblem. Not referenced if ICOMPQ = 0.

GIVCOL (output) INTEGER array, dimension (LDGCOL, 2)
Each pair of numbers indicates a pair of columns to take place in a Givens rotation. Not referenced if ICOMPQ = 0.

LDGCOL (input) INTEGER
The leading dimension of GIVCOL, must be at least N.

GIVNUM (output) DOUBLE PRECISION array, dimension (LDGNUM, 2)
Each number indicates the C or S value to be used in the corresponding Givens rotation. Not referenced if ICOMPQ = 0.

LDGNUM (input) INTEGER
The leading dimension of GIVNUM, must be at least N.

C (output) DOUBLE PRECISION
 C contains garbage if SQRE = 0 and the C-value of a Givens rotation related to the right null space if SQRE = 1.

S (output) DOUBLE PRECISION
 S contains garbage if SQRE = 0 and the S-value of a Givens rotation related to the right null space if SQRE = 1.

INFO (output) INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.

Further Details
 =====

Based on contributions by
 Ming Gu and Huan Ren, Computer Science Division, University of
 California at Berkeley, USA

—————

— dlasd7.f —

```

      SUBROUTINE DLASD7( ICOMPQ, NL, NR, SQRE, K, D, Z, ZW, VF, VFW, VL,
$                      VLW, ALPHA, BETA, DSIGMA, IDX, IDXP, IDXQ,
$                      PERM, GIVPTR, GIVCOL, LDGCOL, GIVNUM, LDGNUM,
$                      C, S, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*  Courant Institute, NAG Ltd., and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      INTEGER          GIVPTR, ICOMPQ, INFO, K, LDGCOL, LDGNUM, NL,
$                      NR, SQRE
      DOUBLE PRECISION ALPHA, BETA, C, S
*
*  ..
*  .. Array Arguments ..
      INTEGER          GIVCOL( LDGCOL, * ), IDX( * ), IDXP( * ),
$                      IDXQ( * ), PERM( * )
      DOUBLE PRECISION D( * ), DSIGMA( * ), GIVNUM( LDGNUM, * ),
$                      VF( * ), VFW( * ), VL( * ), VLW( * ), Z( * ),
$                      ZW( * )
*
*  ..
*
*  =====

```

```

*
*   .. Parameters ..
DOUBLE PRECISION    ZERO, ONE, TWO, EIGHT
PARAMETER            ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0,
$                     EIGHT = 8.0D+0 )
*
*   ..
*   .. Local Scalars ..
*
INTEGER              I, IDXI, IDXJ, IDXJP, J, JP, JPREV, K2, M, N,
$                    NLP1, NLP2
DOUBLE PRECISION     EPS, HLFTOL, TAU, TOL, Z1
*
*   ..
*   .. External Subroutines ..
EXTERNAL              DCOPY, DLAMRG, DROT, XERBLA
*
*   ..
*   .. External Functions ..
DOUBLE PRECISION     DLAMCH, DLAPY2
EXTERNAL              DLAMCH, DLAPY2
*
*   ..
*   .. Intrinsic Functions ..
INTRINSIC             ABS, MAX
*
*   ..
*   .. Executable Statements ..
*
*   Test the input parameters.
*
INFO = 0
N = NL + NR + 1
M = N + SQRE
*
IF( ( ICOMPQ.LT.0 ) .OR. ( ICOMPQ.GT.1 ) ) THEN
    INFO = -1
ELSE IF( NL.LT.1 ) THEN
    INFO = -2
ELSE IF( NR.LT.1 ) THEN
    INFO = -3
ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN
    INFO = -4
ELSE IF( LDGCOL.LT.N ) THEN
    INFO = -22
ELSE IF( LDGNUM.LT.N ) THEN
    INFO = -24
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DLASD7', -INFO )
    RETURN
END IF
*
NLP1 = NL + 1
NLP2 = NL + 2

```

```

      IF( ICOMPQ.EQ.1 ) THEN
        GIVPTR = 0
      END IF

*
*   Generate the first part of the vector Z and move the singular
*   values in the first part of D one position backward.
*
      Z1 = ALPHA*VL( NLP1 )
      VL( NLP1 ) = ZERO
      TAU = VF( NLP1 )
      DO 10 I = NL, 1, -1
        Z( I+1 ) = ALPHA*VL( I )
        VL( I ) = ZERO
        VF( I+1 ) = VF( I )
        D( I+1 ) = D( I )
        IDXQ( I+1 ) = IDXQ( I ) + 1
      10 CONTINUE
      VF( 1 ) = TAU

*
*   Generate the second part of the vector Z.
*
      DO 20 I = NLP2, M
        Z( I ) = BETA*VF( I )
        VF( I ) = ZERO
      20 CONTINUE

*
*   Sort the singular values into increasing order
*
      DO 30 I = NLP2, N
        IDXQ( I ) = IDXQ( I ) + NLP1
      30 CONTINUE

*
*   DSIGMA, IDXC, IDXC, and ZW are used as storage space.
*
      DO 40 I = 2, N
        DSIGMA( I ) = D( IDXQ( I ) )
        ZW( I ) = Z( IDXQ( I ) )
        VFW( I ) = VF( IDXQ( I ) )
        VLW( I ) = VL( IDXQ( I ) )
      40 CONTINUE

*
      CALL DLAMRG( NL, NR, DSIGMA( 2 ), 1, 1, IDX( 2 ) )

*
      DO 50 I = 2, N
        IDXI = 1 + IDX( I )
        D( I ) = DSIGMA( IDXI )
        Z( I ) = ZW( IDXI )
        VF( I ) = VFW( IDXI )
        VL( I ) = VLW( IDXI )
      50 CONTINUE

```

```

*
*   Calculate the allowable deflation tolerance
*
*   EPS = DLAMCH( 'Epsilon' )
*   TOL = MAX( ABS( ALPHA ), ABS( BETA ) )
*   TOL = EIGHT*EIGHT*EPS*MAX( ABS( D( N ) ), TOL )
*
*   There are 2 kinds of deflation -- first a value in the z-vector
*   is small, second two (or more) singular values are very close
*   together (their difference is small).
*
*   If the value in the z-vector is small, we simply permute the
*   array so that the corresponding singular value is moved to the
*   end.
*
*   If two values in the D-vector are close, we perform a two-sided
*   rotation designed to make one of the corresponding z-vector
*   entries zero, and then permute the array so that the deflated
*   singular value is moved to the end.
*
*   If there are multiple singular values then the problem deflates.
*   Here the number of equal singular values are found. As each equal
*   singular value is found, an elementary reflector is computed to
*   rotate the corresponding singular subspace so that the
*   corresponding components of Z are zero in this new basis.
*
*   K = 1
*   K2 = N + 1
*   DO 60 J = 2, N
*       IF( ABS( Z( J ) ) .LE. TOL ) THEN
*
*           Deflate due to small z component.
*
*           K2 = K2 - 1
*           IDXP( K2 ) = J
*           IF( J.EQ.N )
* $             GO TO 100
*           ELSE
*               JPREV = J
*               GO TO 70
*           END IF
*   60 CONTINUE
*   70 CONTINUE
*       J = JPREV
*   80 CONTINUE
*       J = J + 1
*       IF( J.GT.N )
* $         GO TO 90
*       IF( ABS( Z( J ) ) .LE. TOL ) THEN
*

```

```

*      Deflate due to small z component.
*
      K2 = K2 - 1
      IDXP( K2 ) = J
ELSE
*
*      Check if singular values are close enough to allow deflation.
*
      IF( ABS( D( J )-D( JPREV ) ).LE.TOL ) THEN
*
*          Deflation is possible.
*
          S = Z( JPREV )
          C = Z( J )
*
*          Find sqrt(a**2+b**2) without overflow or
*          destructive underflow.
*
          TAU = DLAPY2( C, S )
          Z( J ) = TAU
          Z( JPREV ) = ZERO
          C = C / TAU
          S = -S / TAU
*
*          Record the appropriate Givens rotation
*
          IF( ICOMPQ.EQ.1 ) THEN
              GIVPTR = GIVPTR + 1
              IDXJP = IDXQ( IDX( JPREV )+1 )
              IDXJ = IDXQ( IDX( J )+1 )
              IF( IDXJP.LE.NLP1 ) THEN
                  IDXJP = IDXJP - 1
              END IF
              IF( IDXJ.LE.NLP1 ) THEN
                  IDXJ = IDXJ - 1
              END IF
              GIVCOL( GIVPTR, 2 ) = IDXJP
              GIVCOL( GIVPTR, 1 ) = IDXJ
              GIVNUM( GIVPTR, 2 ) = C
              GIVNUM( GIVPTR, 1 ) = S
          END IF
          CALL DROT( 1, VF( JPREV ), 1, VF( J ), 1, C, S )
          CALL DROT( 1, VL( JPREV ), 1, VL( J ), 1, C, S )
          K2 = K2 - 1
          IDXP( K2 ) = JPREV
          JPREV = J
      ELSE
          K = K + 1
          ZW( K ) = Z( JPREV )
          DSIGMA( K ) = D( JPREV )

```

```

        IDXP( K ) = JPREV
        JPREV = J
    END IF
END IF
GO TO 80
90 CONTINUE
*
*   Record the last singular value.
*
    K = K + 1
    ZW( K ) = Z( JPREV )
    DSIGMA( K ) = D( JPREV )
    IDXP( K ) = JPREV
*
100 CONTINUE
*
*   Sort the singular values into DSIGMA. The singular values which
*   were not deflated go into the first K slots of DSIGMA, except
*   that DSIGMA(1) is treated separately.
*
    DO 110 J = 2, N
        JP = IDXP( J )
        DSIGMA( J ) = D( JP )
        VFW( J ) = VF( JP )
        VLW( J ) = VL( JP )
110 CONTINUE
    IF( ICOMPQ.EQ.1 ) THEN
        DO 120 J = 2, N
            JP = IDXP( J )
            PERM( J ) = IDXQ( IDX( JP )+1 )
            IF( PERM( J ).LE.NLP1 ) THEN
                PERM( J ) = PERM( J ) - 1
            END IF
120 CONTINUE
        END IF
*
*   The deflated singular values go back into the last N - K slots of
*   D.
*
    CALL DCOPY( N-K, DSIGMA( K+1 ), 1, D( K+1 ), 1 )
*
*   Determine DSIGMA(1), DSIGMA(2), Z(1), VF(1), VL(1), VF(M), and
*   VL(M).
*
    DSIGMA( 1 ) = ZERO
    HLFTOL = TOL / TWO
    IF( ABS( DSIGMA( 2 ) ).LE.HLFTOL )
$    DSIGMA( 2 ) = HLFTOL
    IF( M.GT.N ) THEN
        Z( 1 ) = DLAPY2( Z1, Z( M ) )

```

```

      IF( Z( 1 ).LE.TOL ) THEN
        C = ONE
        S = ZERO
        Z( 1 ) = TOL
      ELSE
        C = Z1 / Z( 1 )
        S = -Z( M ) / Z( 1 )
      END IF
      CALL DROT( 1, VF( M ), 1, VF( 1 ), 1, C, S )
      CALL DROT( 1, VL( M ), 1, VL( 1 ), 1, C, S )
    ELSE
      IF( ABS( Z1 ).LE.TOL ) THEN
        Z( 1 ) = TOL
      ELSE
        Z( 1 ) = Z1
      END IF
    END IF
  END IF
*
*   Restore Z, VF, and VL.
*
      CALL DCOPY( K-1, ZW( 2 ), 1, Z( 2 ), 1 )
      CALL DCOPY( N-1, VFW( 2 ), 1, VF( 2 ), 1 )
      CALL DCOPY( N-1, VLW( 2 ), 1, VL( 2 ), 1 )
*
      RETURN
*
*   End of DLASD7
*
      END

```

— LAPACK dlasd7 —

```

(let* ((zero 0.0) (one 1.0) (two 2.0) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 8.0 8.0) eight))
  (defun dlasd7
    (icompq nl nr sqre k d z zw vf vfw vl vlw alpha beta dsigma idx idxp
     idxq perm givptr givcol ldgcol givnum ldgnum c s info)
    (declare (type (simple-array fixnum (*)) givcol perm idxq idxp idx)
              (type (double-float) s c beta alpha)
              (type (simple-array double-float (*)) givnum dsigma vlw vl
                  vfw vf zw z d)
              (type fixnum info ldgnum ldgcol givptr k sqre nr nl
                  icompq))

```



```

(f2cl-lib:with-multi-array-data
  ((d double-float d-%data% d-%offset%)
   (z double-float z-%data% z-%offset%)
   (zw double-float zw-%data% zw-%offset%)
   (vf double-float vf-%data% vf-%offset%)
   (vfw double-float vfw-%data% vfw-%offset%)
   (vl double-float vl-%data% vl-%offset%)
   (vlw double-float vlw-%data% vlw-%offset%)
   (dsigma double-float dsigma-%data% dsigma-%offset%)
   (givnum double-float givnum-%data% givnum-%offset%)
   (idx fixnum idx-%data% idx-%offset%)
   (idxp fixnum idxp-%data% idxp-%offset%)
   (idxq fixnum idxq-%data% idxq-%offset%)
   (perm fixnum perm-%data% perm-%offset%)
   (givcol fixnum givcol-%data% givcol-%offset%))
  (prog ((eps 0.0) (hlftol 0.0) (tau 0.0) (tol 0.0) (z1 0.0) (i 0) (idxi 0)
        (idxj 0) (idxjp 0) (j 0) (jp 0) (jprev 0) (k2 0) (m 0) (n 0)
        (nlp1 0) (nlp2 0))
    (declare (type (double-float) eps hlftol tau tol z1)
              (type fixnum i idxi idxj idxjp j jp jprev k2 m n
                        nlp1 nlp2))

    (setf info 0)
    (setf n (f2cl-lib:int-add nl nr 1))
    (setf m (f2cl-lib:int-add n sqre))
    (cond
      ((or (< icompq 0) (> icompq 1))
       (setf info -1))
      ((< nl 1)
       (setf info -2))
      ((< nr 1)
       (setf info -3))
      ((or (< sqre 0) (> sqre 1))
       (setf info -4))
      ((< ldgcol n)
       (setf info -22))
      ((< ldgnum n)
       (setf info -24)))
    (cond
      ((/= info 0)
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DLASD7" (f2cl-lib:int-sub info))
       (go end_label)))
    (setf nlp1 (f2cl-lib:int-add nl 1))
    (setf nlp2 (f2cl-lib:int-add nl 2))
    (cond
      ((= icompq 1)
       (setf givptr 0)))
    (setf z1
      (* alpha (f2cl-lib:fref vl-%data% (nlp1) ((1 *)) vl-%offset%)))

```

```

(setf (f2cl-lib:fref vl-%data% (nlp1) ((1 *)) vl-%offset%) zero)
(setf tau (f2cl-lib:fref vf-%data% (nlp1) ((1 *)) vf-%offset%))
(f2cl-lib:fdo (i n1 (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
  (> i 1) nil)
(tagbody
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-add i 1))
    ((1 *))
    z-%offset%)
    (* alpha
      (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%)))
  (setf (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%) zero)
  (setf (f2cl-lib:fref vf-%data%
    ((f2cl-lib:int-add i 1))
    ((1 *))
    vf-%offset%)
    (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%))
  (setf (f2cl-lib:fref d-%data%
    ((f2cl-lib:int-add i 1))
    ((1 *))
    d-%offset%)
    (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref idxq-%data%
    ((f2cl-lib:int-add i 1))
    ((1 *))
    idxq-%offset%)
    (f2cl-lib:int-add
      (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
      1))))
(setf (f2cl-lib:fref vf-%data% (1) ((1 *)) vf-%offset%) tau)
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
  (> i m) nil)
(tagbody
  (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
    (* beta (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%)))
  (setf (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%) zero))
(f2cl-lib:fdo (i nlp2 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
    (f2cl-lib:int-add
      (f2cl-lib:fref idxq-%data% (i) ((1 *)) idxq-%offset%)
      nlp1))))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
    (f2cl-lib:fref d-%data%
      ((f2cl-lib:fref idxq (i) ((1 *)))
      ((1 *)))

```

```

                                d-%offset%))
  (setf (f2cl-lib:fref zw-%data% (i) ((1 *)) zw-%offset%)
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:fref idxq (i) ((1 *)))
                         ((1 *))
                         z-%offset%))
        (setf (f2cl-lib:fref vfw-%data% (i) ((1 *)) vfw-%offset%)
              (f2cl-lib:fref vf-%data%
                              ((f2cl-lib:fref idxq (i) ((1 *)))
                               ((1 *))
                               vf-%offset%))
              (setf (f2cl-lib:fref vlw-%data% (i) ((1 *)) vlw-%offset%)
                    (f2cl-lib:fref vl-%data%
                                    ((f2cl-lib:fref idxq (i) ((1 *)))
                                     ((1 *))
                                     vl-%offset%))))))
  (dlamrg nl nr (f2cl-lib:array-slice dsigma double-float (2) ((1 *))) 1
    1 (f2cl-lib:array-slice idx fixnum (2) ((1 *))))
  (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
    (> i n) nil)
  (tagbody
    (setf idxi
          (f2cl-lib:int-add 1
                            (f2cl-lib:fref idx-%data%
                                              (i)
                                              ((1 *))
                                              idx-%offset%)))
    (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          (f2cl-lib:fref dsigma-%data%
                          (idxi)
                          ((1 *))
                          dsigma-%offset%))
    (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
          (f2cl-lib:fref zw-%data% (idxi) ((1 *)) zw-%offset%))
    (setf (f2cl-lib:fref vf-%data% (i) ((1 *)) vf-%offset%)
          (f2cl-lib:fref vfw-%data% (idxi) ((1 *)) vfw-%offset%))
    (setf (f2cl-lib:fref vl-%data% (i) ((1 *)) vl-%offset%)
          (f2cl-lib:fref vlw-%data% (idxi) ((1 *)) vlw-%offset%)))
  (setf eps (dlamch "Epsilon"))
  (setf tol (max (abs alpha) (abs beta)))
  (setf tol
    (* eight
      eight
      eps
      (max (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
              tol))))
  (setf k 1)
  (setf k2 (f2cl-lib:int-add n 1))
  (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
    (> j n) nil)

```

```

(tagbody
  (cond
    ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
      (setf k2 (f2cl-lib:int-sub k2 1))
      (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j)
      (if (= j n) (go label100)))
    (t
      (setf jprev j)
      (go label70))))
label70
  (setf j jprev)
label80
  (setf j (f2cl-lib:int-add j 1))
  (if (> j n) (go label90))
  (cond
    ((<= (abs (f2cl-lib:fref z (j) ((1 *)))) tol)
      (setf k2 (f2cl-lib:int-sub k2 1))
      (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%) j))
    (t
      (cond
        ((<=
          (abs
            (+ (f2cl-lib:fref d (j) ((1 *)))
              (- (f2cl-lib:fref d (jprev) ((1 *))))))
          tol)
          (setf s (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
          (setf c (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%))
          (setf tau (dlapy2 c s))
          (setf (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%) tau)
          (setf (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%) zero)
          (setf c (/ c tau))
          (setf s (/ (- s) tau))
          (cond
            ((= icompq 1)
              (setf givptr (f2cl-lib:int-add givptr 1))
              (setf idxjp
                (f2cl-lib:fref idxq-%data%
                  ((f2cl-lib:int-add
                    (f2cl-lib:fref idx (jprev) ((1 *))
                    1))
                  ((1 *))
                  idxq-%offset%))
                (setf idxj
                  (f2cl-lib:fref idxq-%data%
                    ((f2cl-lib:int-add
                      (f2cl-lib:fref idx (j) ((1 *))
                      1))
                    ((1 *))
                    idxq-%offset%))
                  (cond

```

```

      ((<= idxjp nlp1)
        (setf idxjp (f2cl-lib:int-sub idxjp 1))))
    (cond
      ((<= idxj nlp1)
        (setf idxj (f2cl-lib:int-sub idxj 1))))
    (setf (f2cl-lib:fref givcol-%data%
      (givptr 2)
      ((1 ldgcol) (1 *))
      givcol-%offset%)
      idxjp)
    (setf (f2cl-lib:fref givcol-%data%
      (givptr 1)
      ((1 ldgcol) (1 *))
      givcol-%offset%)
      idxj)
    (setf (f2cl-lib:fref givnum-%data%
      (givptr 2)
      ((1 ldgnum) (1 *))
      givnum-%offset%)
      c)
    (setf (f2cl-lib:fref givnum-%data%
      (givptr 1)
      ((1 ldgnum) (1 *))
      givnum-%offset%)
      s)))
  (drot 1 (f2cl-lib:array-slice vf double-float (jprev) ((1 *))) 1
    (f2cl-lib:array-slice vf double-float (j) ((1 *))) 1 c s)
  (drot 1 (f2cl-lib:array-slice vl double-float (jprev) ((1 *))) 1
    (f2cl-lib:array-slice vl double-float (j) ((1 *))) 1 c s)
  (setf k2 (f2cl-lib:int-sub k2 1))
  (setf (f2cl-lib:fref idxp-%data% (k2) ((1 *)) idxp-%offset%)
    jprev)
  (setf jprev j))
  (t
    (setf k (f2cl-lib:int-add k 1))
    (setf (f2cl-lib:fref zw-%data% (k) ((1 *)) zw-%offset%)
      (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
    (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
      (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
    (setf jprev j))))
  (go label80)
label190
  (setf k (f2cl-lib:int-add k 1))
  (setf (f2cl-lib:fref zw-%data% (k) ((1 *)) zw-%offset%)
    (f2cl-lib:fref z-%data% (jprev) ((1 *)) z-%offset%))
  (setf (f2cl-lib:fref dsigma-%data% (k) ((1 *)) dsigma-%offset%)
    (f2cl-lib:fref d-%data% (jprev) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref idxp-%data% (k) ((1 *)) idxp-%offset%) jprev)
label1100

```

```

(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
    (setf (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
      (f2cl-lib:fref d-%data% (jp) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref vfw-%data% (j) ((1 *)) vfw-%offset%)
      (f2cl-lib:fref vf-%data% (jp) ((1 *)) vf-%offset%))
    (setf (f2cl-lib:fref vlw-%data% (j) ((1 *)) vlw-%offset%)
      (f2cl-lib:fref vl-%data% (jp) ((1 *)) vl-%offset%)))
  (cond
    ((= icompq 1)
      (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
        (> j n) nil)
        (tagbody
          (setf jp (f2cl-lib:fref idxp-%data% (j) ((1 *)) idxp-%offset%))
          (setf (f2cl-lib:fref perm-%data% (j) ((1 *)) perm-%offset%)
            (f2cl-lib:fref idxq-%data%
              ((f2cl-lib:int-add
                (f2cl-lib:fref idx (jp) ((1 *)))
                1))
              ((1 *))
              idxq-%offset%)))
          (cond
            ((<= (f2cl-lib:fref perm (j) ((1 *))) nlp1)
              (setf (f2cl-lib:fref perm-%data% (j) ((1 *)) perm-%offset%)
                (f2cl-lib:int-sub
                  (f2cl-lib:fref perm-%data%
                    (j)
                    ((1 *))
                    perm-%offset%)
                  1))))))
      (dcopy (f2cl-lib:int-sub n k)
        (f2cl-lib:array-slice dsigma double-float ((+ k 1)) ((1 *))) 1
        (f2cl-lib:array-slice d double-float ((+ k 1)) ((1 *))) 1)
      (setf (f2cl-lib:fref dsigma-%data% (1) ((1 *)) dsigma-%offset%) zero)
      (setf hlftol (/ tol two))
      (if
        (<= (abs (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%))
          hlftol)
        (setf (f2cl-lib:fref dsigma-%data% (2) ((1 *)) dsigma-%offset%)
          hlftol))
      (cond
        (> m n)
        (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
          (dlapy2 z1 (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)))
        (cond
          (<= (f2cl-lib:fref z (1) ((1 *))) tol)
          (setf c one)
          (setf s zero)

```

```
(setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
(t
  (setf c (/ z1 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
  (setf s
    (/ (- (f2cl-lib:fref z-%data% (m) ((1 *)) z-%offset%)
          (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))))))
(drot 1 (f2cl-lib:array-slice vf double-float (m) ((1 *))) 1
        (f2cl-lib:array-slice zf double-float (1) ((1 *))) 1 c s)
(drot 1 (f2cl-lib:array-slice vl double-float (m) ((1 *))) 1
        (f2cl-lib:array-slice vl double-float (1) ((1 *))) 1 c s))
(t
  (cond
    ((<= (abs z1) tol)
     (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) tol))
    (t
     (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) z1))))
(dcopy (f2cl-lib:int-sub k 1)
      (f2cl-lib:array-slice zw double-float (2) ((1 *))) 1
      (f2cl-lib:array-slice z double-float (2) ((1 *))) 1)
(dcopy (f2cl-lib:int-sub n 1)
      (f2cl-lib:array-slice vfw double-float (2) ((1 *))) 1
      (f2cl-lib:array-slice vf double-float (2) ((1 *))) 1)
(dcopy (f2cl-lib:int-sub n 1)
      (f2cl-lib:array-slice vlw double-float (2) ((1 *))) 1
      (f2cl-lib:array-slice vl double-float (2) ((1 *))) 1)
end_label
(return
  (values nil
          nil
          nil
          k
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          nil
          givptr
          nil
          nil
          nil)
```

```

nil
c
s
info))))))

```

dlasd8 LAPACK

— dlasd8.input —

```

)set break resume
)sys rm -f dlasd8.output
)spool dlasd8.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasd8.help —

```

=====
dlasd8 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASD8 - the square roots of the roots of the secular equation,

SYNOPSIS

```

SUBROUTINE DLASD8( ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDDIFR, DSIGMA,
                  WORK, INFO )

```

INTEGER ICOMPQ, INFO, K, LDDIFR

DOUBLE PRECISION D(*), DIFL(*), DIFR(LDDIFR, *),
 DSIGMA(*), VF(*), VL(*), WORK(*), Z(*)

Purpose

=====

DLASD8 finds the square roots of the roots of the secular equation, as defined by the values in DSIGMA and Z. It makes the appropriate calls to DLASD4, and stores, for each element in D, the distance to its two nearest poles (elements in DSIGMA). It also updates the arrays VF and VL, the first and last components of all the right singular vectors of the original bidiagonal matrix.

DLASD8 is called from DLASD6.

Arguments

=====

ICOMPQ (input) INTEGER

Specifies whether singular vectors are to be computed in factored form in the calling routine:

= 0: Compute singular values only.

= 1: Compute singular vectors in factored form as well.

K (input) INTEGER

The number of terms in the rational function to be solved by DLASD4. $K \geq 1$.

D (output) DOUBLE PRECISION array, dimension (K)

On output, D contains the updated singular values.

Z (input) DOUBLE PRECISION array, dimension (K)

The first K elements of this array contain the components of the deflation-adjusted updating row vector.

VF (input/output) DOUBLE PRECISION array, dimension (K)

On entry, VF contains information passed through DBEDE8.

On exit, VF contains the first K components of the first components of all right singular vectors of the bidiagonal matrix.

VL (input/output) DOUBLE PRECISION array, dimension (K)

On entry, VL contains information passed through DBEDE8.

On exit, VL contains the first K components of the last components of all right singular vectors of the bidiagonal matrix.

DIFL (output) DOUBLE PRECISION array, dimension (K)

On exit, $DIFL(I) = D(I) - DSIGMA(I)$.

DIFR (output) DOUBLE PRECISION array,

dimension (LDDIFR, 2) if ICOMPQ = 1 and

dimension (K) if ICOMPQ = 0.

On exit, $\text{DIFR}(I,1) = D(I) - \text{DSIGMA}(I+1)$, $\text{DIFR}(K,1)$ is not defined and will not be referenced.

If $\text{ICOMPQ} = 1$, $\text{DIFR}(1:K,2)$ is an array containing the normalizing factors for the right singular vector matrix.

LDDIFR (input) INTEGER

The leading dimension of **DIFR**, must be at least K .

DSIGMA (input) DOUBLE PRECISION array, dimension (K)

The first K elements of this array contain the old roots of the deflated updating problem. These are the poles of the secular equation.

WORK (workspace) DOUBLE PRECISION array, dimension at least $3 * K$

INFO (output) INTEGER

= 0: successful exit.

< 0: if $\text{INFO} = -i$, the i -th argument had an illegal value.

> 0: if $\text{INFO} = 1$, an singular value did not converge

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of California at Berkeley, USA

— dlasd8.f —

```

SUBROUTINE DLASD8( ICOMPQ, K, D, Z, VF, VL, DIFL, DIFR, LDDIFR,
$                DSIGMA, WORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Oak Ridge National Lab, Argonne National Lab,
*    Courant Institute, NAG Ltd., and Rice University
*    June 30, 1999
*
*    .. Scalar Arguments ..
INTEGER          ICOMPQ, INFO, K, LDDIFR
*
*    .. Array Arguments ..
DOUBLE PRECISION D( * ), DIFL( * ), DIFR( LDDIFR, * ),
$                DSIGMA( * ), VF( * ), VL( * ), WORK( * ),
$                Z( * )
*
*    ..

```

```

*
* =====
*
* .. Parameters ..
DOUBLE PRECISION    ONE
PARAMETER            ( ONE = 1.0D+0 )
*
* ..
* .. Local Scalars ..
INTEGER              I, IWK1, IWK2, IWK2I, IWK3, IWK3I, J
DOUBLE PRECISION     DIFLJ, DIFRJ, DJ, DSIGJ, DSIGJP, RHO, TEMP
*
* ..
* .. External Subroutines ..
EXTERNAL             DCOPY, DLASCL, DLASD4, DLASET, XERBLA
*
* ..
* .. External Functions ..
DOUBLE PRECISION     DDOT, DLAMC3, DNRM2
EXTERNAL             DDOT, DLAMC3, DNRM2
*
* ..
* .. Intrinsic Functions ..
INTRINSIC             ABS, SIGN, SQRT
*
* ..
* .. Executable Statements ..
*
* Test the input parameters.
*
* INFO = 0
*
* IF( ( ICOMPQ.LT.0 ) .OR. ( ICOMPQ.GT.1 ) ) THEN
*   INFO = -1
* ELSE IF( K.LT.1 ) THEN
*   INFO = -2
* ELSE IF( LDDIFR.LT.K ) THEN
*   INFO = -9
* END IF
* IF( INFO.NE.0 ) THEN
*   CALL XERBLA( 'DLASD8', -INFO )
*   RETURN
* END IF
*
* Quick return if possible
*
* IF( K.EQ.1 ) THEN
*   D( 1 ) = ABS( Z( 1 ) )
*   DIFL( 1 ) = D( 1 )
*   IF( ICOMPQ.EQ.1 ) THEN
*     DIFL( 2 ) = ONE
*     DIFR( 1, 2 ) = ONE
*   END IF
*   RETURN
* END IF

```

```

*
*   Modify values DSIGMA(i) to make sure all DSIGMA(i)-DSIGMA(j) can
*   be computed with high relative accuracy (barring over/underflow).
*   This is a problem on machines without a guard digit in
*   add/subtract (Cray XMP, Cray YMP, Cray C 90 and Cray 2).
*   The following code replaces DSIGMA(I) by 2*DSIGMA(I)-DSIGMA(I),
*   which on any of these machines zeros out the bottommost
*   bit of DSIGMA(I) if it is 1; this makes the subsequent
*   subtractions DSIGMA(I)-DSIGMA(J) unproblematic when cancellation
*   occurs. On binary machines with a guard digit (almost all
*   machines) it does not change DSIGMA(I) at all. On hexadecimal
*   and decimal machines with a guard digit, it slightly
*   changes the bottommost bits of DSIGMA(I). It does not account
*   for hexadecimal or decimal machines without guard digits
*   (we know of none). We use a subroutine call to compute
*   2*DLAMBDA(I) to prevent optimizing compilers from eliminating
*   this code.
*
      DO 10 I = 1, K
        DSIGMA( I ) = DLAMC3( DSIGMA( I ), DSIGMA( I ) ) - DSIGMA( I )
10    CONTINUE
*
*   Book keeping.
*
      IWK1 = 1
      IWK2 = IWK1 + K
      IWK3 = IWK2 + K
      IWK2I = IWK2 - 1
      IWK3I = IWK3 - 1
*
*   Normalize Z.
*
      RHO = DNRM2( K, Z, 1 )
      CALL DLASCL( 'G', 0, 0, RHO, ONE, K, 1, Z, K, INFO )
      RHO = RHO*RHO
*
*   Initialize WORK(IWK3).
*
      CALL DLASET( 'A', K, 1, ONE, ONE, WORK( IWK3 ), K )
*
*   Compute the updated singular values, the arrays DIFL, DIFR,
*   and the updated Z.
*
      DO 40 J = 1, K
        CALL DLASD4( K, J, DSIGMA, Z, WORK( IWK1 ), RHO, D( J ),
$           WORK( IWK2 ), INFO )
*
*       If the root finder fails, the computation is terminated.
*
      IF( INFO.NE.0 ) THEN

```

```

        RETURN
    END IF
    WORK( IWK3I+J ) = WORK( IWK3I+J )*WORK( J )*WORK( IWK2I+J )
    DIFL( J ) = -WORK( J )
    DIFR( J, 1 ) = -WORK( J+1 )
    DO 20 I = 1, J - 1
        WORK( IWK3I+I ) = WORK( IWK3I+I )*WORK( I )*
$           WORK( IWK2I+I ) / ( DSIGMA( I )-
$           DSIGMA( J ) ) / ( DSIGMA( I )+
$           DSIGMA( J ) )
20    CONTINUE
    DO 30 I = J + 1, K
        WORK( IWK3I+I ) = WORK( IWK3I+I )*WORK( I )*
$           WORK( IWK2I+I ) / ( DSIGMA( I )-
$           DSIGMA( J ) ) / ( DSIGMA( I )+
$           DSIGMA( J ) )
30    CONTINUE
40    CONTINUE
*
*      Compute updated Z.
*
    DO 50 I = 1, K
        Z( I ) = SIGN( SQRT( ABS( WORK( IWK3I+I ) ) ), Z( I ) )
50    CONTINUE
*
*      Update VF and VL.
*
    DO 80 J = 1, K
        DIFLJ = DIFL( J )
        DJ = D( J )
        DSIGJ = -DSIGMA( J )
        IF( J.LT.K ) THEN
            DIFRJ = -DIFR( J, 1 )
            DSIGJP = -DSIGMA( J+1 )
        END IF
        WORK( J ) = -Z( J ) / DIFLJ / ( DSIGMA( J )+DJ )
        DO 60 I = 1, J - 1
            WORK( I ) = Z( I ) / ( DLAMC3( DSIGMA( I ), DSIGJ )-DIFLJ )
$           / ( DSIGMA( I )+DJ )
60    CONTINUE
        DO 70 I = J + 1, K
            WORK( I ) = Z( I ) / ( DLAMC3( DSIGMA( I ), DSIGJP )+DIFRJ )
$           / ( DSIGMA( I )+DJ )
70    CONTINUE
        TEMP = DNRM2( K, WORK, 1 )
        WORK( IWK2I+J ) = DDOT( K, WORK, 1, VF, 1 ) / TEMP
        WORK( IWK3I+J ) = DDOT( K, WORK, 1, VL, 1 ) / TEMP
        IF( ICOMPQ.EQ.1 ) THEN
            DIFR( J, 2 ) = TEMP
        END IF
    END IF

```

```

      80 CONTINUE
*
      CALL DCOPY( K, WORK( IWK2 ), 1, VF, 1 )
      CALL DCOPY( K, WORK( IWK3 ), 1, VL, 1 )
*
      RETURN
*
*      End of DLASD8
*
      END

```

— LAPACK dlasd8 —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dlasd8 (icompq k d z vf vl difl difr lddifr dsigma work info)
    (declare (type (simple-array double-float (*)) work dsigma difr difl vl vf z d)
      (type fixnum info lddifr k icompq))
    (f2cl-lib:with-multi-array-data
      ((d double-float d-%data% d-%offset%)
       (z double-float z-%data% z-%offset%)
       (vf double-float vf-%data% vf-%offset%)
       (vl double-float vl-%data% vl-%offset%)
       (difl double-float difl-%data% difl-%offset%)
       (difr double-float difr-%data% difr-%offset%)
       (dsigma double-float dsigma-%data% dsigma-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((diflj 0.0) (difrj 0.0) (dj 0.0) (dsigj 0.0) (dsigjp 0.0)
             (rho 0.0) (temp 0.0) (i 0) (iwk1 0) (iwk2 0) (iwk2i 0) (iwk3 0)
             (iwk3i 0) (j 0))
        (declare (type (double-float) diflj difrj dj dsigj dsigjp rho temp)
          (type fixnum i iwkl iwkl2 iwkl2i iwkl3 iwkl3i j))
        (setf info 0)
        (cond
          ((or (< icompq 0) (> icompq 1))
            (setf info -1))
          ((< k 1)
            (setf info -2))
          ((< lddifr k)
            (setf info -9)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DLASD8" (f2cl-lib:int-sub info))
            (go end_label)))

```

```

(cond
  ((= k 1)
    (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
          (abs (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
    (setf (f2cl-lib:fref difl-%data% (1) ((1 *)) difl-%offset%)
          (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%))
    (cond
      ((= icompq 1)
        (setf (f2cl-lib:fref difl-%data% (2) ((1 *)) difl-%offset%) one)
        (setf (f2cl-lib:fref difr-%data%
                          (1 2)
                          ((1 lddifr) (1 *))
                          difr-%offset%)
              one)))
      (go end_label)))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i k) nil)
    (tagbody
      (setf (f2cl-lib:fref dsigma-%data% (i) ((1 *)) dsigma-%offset%)
            (-
              (multiple-value-bind (ret-val var-0 var-1)
                (dlamc3
                  (f2cl-lib:fref dsigma-%data%
                                (i)
                                ((1 *))
                                dsigma-%offset%)
                  (f2cl-lib:fref dsigma-%data%
                                (i)
                                ((1 *))
                                dsigma-%offset%)))
              (declare (ignore))
              (setf (f2cl-lib:fref dsigma-%data%
                                (i)
                                ((1 *))
                                dsigma-%offset%)
                    var-0)
              (setf (f2cl-lib:fref dsigma-%data%
                                (i)
                                ((1 *))
                                dsigma-%offset%)
                    var-1)
              ret-val)
            (f2cl-lib:fref dsigma-%data%
                          (i)
                          ((1 *))
                          dsigma-%offset%))))))
  (setf iwk1 1)
  (setf iwk2 (f2cl-lib:int-add iwk1 k))
  (setf iwk3 (f2cl-lib:int-add iwk2 k))
  (setf iwk2i (f2cl-lib:int-sub iwk2 1))

```

```

(setf iwk3i (f2cl-lib:int-sub iwk3 1))
(setf rho (dnrm2 k z 1))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlascl "G" 0 0 rho one k 1 z k info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8))
  (setf info var-9))
(setf rho (* rho rho))
(dlaset "A" k 1 one one
  (f2cl-lib:array-slice work double-float (iwk3) ((1 *))) k)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
    (dlasd4 k j dsigma z
      (f2cl-lib:array-slice work double-float (iwk1) ((1 *))) rho
      (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
      (f2cl-lib:array-slice work double-float (iwk2) ((1 *))) info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
    (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) var-6)
    (setf info var-8))
  (cond
    ((/= info 0)
     (go end_label)))
  (setf (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add iwk3i j))
    ((1 *))
    work-%offset%)
    (*
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add iwk3i j))
        ((1 *))
        work-%offset%)
      (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add iwk2i j))
        ((1 *))
        work-%offset%)))
  (setf (f2cl-lib:fref difl-%data% (j) ((1 *)) difl-%offset%)
    (- (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)))
  (setf (f2cl-lib:fref difr-%data%
    (j 1)
    ((1 lddifr) (1 *))
    difr-%offset%)
    (-
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add j 1))
        ((1 *))

```



```

                                work-%offset%)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-add iwk3i i))
      ((1 *))
      work-%offset%))
      (/
        (/
          (*
            (f2cl-lib:fref work-%data%
              ((f2cl-lib:int-add iwk3i i))
              ((1 *))
              work-%offset%)
            (f2cl-lib:fref work-%data%
              (i)
              ((1 *))
              work-%offset%)
            (f2cl-lib:fref work-%data%
              ((f2cl-lib:int-add iwk2i i))
              ((1 *))
              work-%offset%)))
          (-
            (f2cl-lib:fref dsigma-%data%
              (i)
              ((1 *))
              dsigma-%offset%)
            (f2cl-lib:fref dsigma-%data%
              (j)
              ((1 *))
              dsigma-%offset%)))
          (+
            (f2cl-lib:fref dsigma-%data%
              (i)
              ((1 *))
              dsigma-%offset%)
            (f2cl-lib:fref dsigma-%data%
              (j)
              ((1 *))
              dsigma-%offset%))))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
  ((> i k) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-add iwk3i i))
      ((1 *))
      work-%offset%)
      (/
        (/

```

```

(*
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add iwk3i i))
    ((1 *))
    work-%offset%)
  (f2cl-lib:fref work-%data%
    (i)
    ((1 *))
    work-%offset%)
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add iwk2i i))
    ((1 *))
    work-%offset%))
(-
  (f2cl-lib:fref dsigma-%data%
    (i)
    ((1 *))
    dsigma-%offset%)
  (f2cl-lib:fref dsigma-%data%
    (j)
    ((1 *))
    dsigma-%offset%)))
(+
  (f2cl-lib:fref dsigma-%data%
    (i)
    ((1 *))
    dsigma-%offset%)
  (f2cl-lib:fref dsigma-%data%
    (j)
    ((1 *))
    dsigma-%offset%))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i k) nil)
(tagbody
  (setf (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
    (f2cl-lib:sign
      (f2cl-lib:fsqrt
        (abs
          (f2cl-lib:fref work-%data%
            ((f2cl-lib:int-add iwk3i i))
            ((1 *))
            work-%offset%)))
        (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
(tagbody
  (setf diflj (f2cl-lib:fref difl-%data% (j) ((1 *)) difl-%offset%))
  (setf dj (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
  (setf dsigj
    (-

```

```

        (f2cl-lib:fref dsigma-%data%
          (j)
          ((1 *))
          dsigma-%offset%)))
(cond
  ((< j k)
   (setf difrj
     (-
      (f2cl-lib:fref difr-%data%
        (j 1)
        ((1 lddifr) (1 *))
        difr-%offset%)))

   (setf dsigjp
     (-
      (f2cl-lib:fref dsigma-%data%
        ((f2cl-lib:int-add j 1))
        ((1 *))
        dsigma-%offset%))))))
(setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
  (/
    (/ (- (f2cl-lib:fref z-%data% (j) ((1 *)) z-%offset%)
          diflj)
      (+
        (f2cl-lib:fref dsigma-%data% (j) ((1 *)) dsigma-%offset%)
        dj)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
    (/
      (/ (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
        (-
          (multiple-value-bind (ret-val var-0 var-1)
            (dlamc3
              (f2cl-lib:fref dsigma-%data%
                (i)
                ((1 *))
                dsigma-%offset%)
              dsigj)
            (declare (ignore))
            (setf (f2cl-lib:fref dsigma-%data%
              (i)
              ((1 *))
              dsigma-%offset%)
                var-0)
            (setf dsigj var-1)
            ret-val)
          diflj))
      (+
        (f2cl-lib:fref dsigma-%data%

```

```

                                (i)
                                ((1 *))
                                dsigma-%offset%)
                                dj))))
(f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
              (> i k) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
        (/
          (f2cl-lib:fref z-%data% (i) ((1 *)) z-%offset%)
          (+
            (multiple-value-bind (ret-val var-0 var-1)
              (dlamc3
                (f2cl-lib:fref dsigma-%data%
                              (i)
                              ((1 *))
                              dsigma-%offset%)
                dsigjp)
              (declare (ignore))
              (setf (f2cl-lib:fref dsigma-%data%
                                  (i)
                                  ((1 *))
                                  dsigma-%offset%)
                    var-0)
              (setf dsigjp var-1)
              ret-val)
            difrj))
          (+
            (f2cl-lib:fref dsigma-%data%
                          (i)
                          ((1 *))
                          dsigma-%offset%)
            dj))))
(setf temp (dnrm2 k work 1))
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add iwk2i j))
                    ((1 *))
                    work-%offset%)
      (/ (ddot k work 1 vf 1) temp))
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add iwk3i j))
                    ((1 *))
                    work-%offset%)
      (/ (ddot k work 1 vl 1) temp))
(cond
  ((= icompq 1)
   (setf (f2cl-lib:fref difr-%data%
                       (j 2)
                       ((1 lddifr) (1 *))
                       difr-%offset%)

```

```

                                temp))))
      (dcopy k (f2cl-lib:array-slice work double-float (iwk2) ((1 *))) 1 vf 1)
      (dcopy k (f2cl-lib:array-slice work double-float (iwk3) ((1 *))) 1 vl 1)
end_label
      (return (values nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dlasda LAPACK

— dlasda.input —

```

)set break resume
)sys rm -f dlasda.output
)spool dlasda.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasda.help —

```
=====
dlasda examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASDA - divide and conquer approach, DLASDA computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E, where $M = N + \text{SQRE}$

SYNOPSIS

```

SUBROUTINE DLASDA( ICOMPQ, SMLSIZ, N, SQRE, D, E, U, LDU, VT, K, DIFL,
                  DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL, PERM,
                  GIVNUM, C, S, WORK, IWORK, INFO )

```

```

      INTEGER      ICOMPQ, INFO, LDGCOL, LDU, N, SMLSIZ, SQRE

```

```

      INTEGER      GIVCOL( LDGCOL, * ), GIVPTR( * ), IWORK( * ), K( *
                      ), PERM( LDGCOL, * )

      DOUBLE       PRECISION C( * ), D( * ), DIFL( LDU, * ), DIFR( LDU,
                      * ), E( * ), GIVNUM( LDU, * ), POLES( LDU, * ), S( *
                      ), U( LDU, * ), VT( LDU, * ), WORK( * ), Z( LDU, * )

```

Purpose

=====

Using a divide and conquer approach, DLASDA computes the singular value decomposition (SVD) of a real upper bidiagonal N -by- M matrix B with diagonal D and offdiagonal E , where $M = N + \text{SQRE}$. The algorithm computes the singular values in the SVD $B = U * S * VT$. The orthogonal matrices U and VT are optionally computed in compact form.

A related subroutine, DLASDO, computes the singular values and the singular vectors in explicit form.

Arguments

=====

ICOMPQ (input) INTEGER

Specifies whether singular vectors are to be computed in compact form, as follows
 = 0: Compute singular values only.
 = 1: Compute singular vectors of upper bidiagonal matrix in compact form.

SMLSIZ (input) INTEGER

The maximum size of the subproblems at the bottom of the computation tree.

N (input) INTEGER

The row dimension of the upper bidiagonal matrix. This is also the dimension of the main diagonal array D .

SQRE (input) INTEGER

Specifies the column dimension of the bidiagonal matrix.
 = 0: The bidiagonal matrix has column dimension $M = N$;
 = 1: The bidiagonal matrix has column dimension $M = N + 1$.

D (input/output) DOUBLE PRECISION array, dimension (N)

On entry D contains the main diagonal of the bidiagonal matrix. On exit D , if $\text{INFO} = 0$, contains its singular values.

E (input) DOUBLE PRECISION array, dimension (M-1)

Contains the subdiagonal entries of the bidiagonal matrix.

On exit, E has been destroyed.

- U (output) DOUBLE PRECISION array,
dimension (LDU, SMLSIZ) if ICOMPQ = 1, and not referenced
if ICOMPQ = 0. If ICOMPQ = 1, on exit, U contains the left
singular vector matrices of all subproblems at the bottom
level.
- LDU (input) INTEGER, LDU = > N.
The leading dimension of arrays U, VT, DIFL, DIFR, POLES,
GIVNUM, and Z.
- VT (output) DOUBLE PRECISION array,
dimension (LDU, SMLSIZ+1) if ICOMPQ = 1, and not referenced
if ICOMPQ = 0. If ICOMPQ = 1, on exit, VT' contains the right
singular vector matrices of all subproblems at the bottom
level.
- K (output) INTEGER array,
dimension (N) if ICOMPQ = 1 and dimension 1 if ICOMPQ = 0.
If ICOMPQ = 1, on exit, K(I) is the dimension of the I-th
secular equation on the computation tree.
- DIFL (output) DOUBLE PRECISION array, dimension (LDU, NLVL),
where NLVL = floor(log₂ (N/SMLSIZ)).
- DIFR (output) DOUBLE PRECISION array,
dimension (LDU, 2 * NLVL) if ICOMPQ = 1 and
dimension (N) if ICOMPQ = 0.
If ICOMPQ = 1, on exit, DIFL(1:N, I) and DIFR(1:N, 2 * I - 1)
record distances between singular values on the I-th
level and singular values on the (I -1)-th level, and
DIFR(1:N, 2 * I) contains the normalizing factors for
the right singular vector matrix. See DLASD8 for details.
- Z (output) DOUBLE PRECISION array,
dimension (LDU, NLVL) if ICOMPQ = 1 and
dimension (N) if ICOMPQ = 0.
The first K elements of Z(1, I) contain the components of
the deflation-adjusted updating row vector for subproblems
on the I-th level.
- POLES (output) DOUBLE PRECISION array,
dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced
if ICOMPQ = 0. If ICOMPQ = 1, on exit, POLES(1, 2*I - 1) and
POLES(1, 2*I) contain the new and old singular values
involved in the secular equations on the I-th level.
- GIVPTR (output) INTEGER array,
dimension (N) if ICOMPQ = 1, and not referenced if

ICOMPQ = 0. If ICOMPQ = 1, on exit, GIVPTR(I) records the number of Givens rotations performed on the I-th problem on the computation tree.

GIVCOL (output) INTEGER array,
dimension (LDGCOL, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVCOL(1, 2 * I - 1) and GIVCOL(1, 2 * I) record the locations of Givens rotations performed on the I-th level on the computation tree.

LDGCOL (input) INTEGER, LDGCOL = > N.
The leading dimension of arrays GIVCOL and PERM.

PERM (output) INTEGER array,
dimension (LDGCOL, NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, PERM(1, I) records permutations done on the I-th level of the computation tree.

GIVNUM (output) DOUBLE PRECISION array,
dimension (LDU, 2 * NLVL) if ICOMPQ = 1, and not referenced if ICOMPQ = 0. If ICOMPQ = 1, on exit, for each I, GIVNUM(1, 2 * I - 1) and GIVNUM(1, 2 * I) record the C- and S-values of Givens rotations performed on the I-th level on the computation tree.

C (output) DOUBLE PRECISION array,
dimension (N) if ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, C(I) contains the C-value of a Givens rotation related to the right null space of the I-th subproblem.

S (output) DOUBLE PRECISION array, dimension (N) if ICOMPQ = 1, and dimension 1 if ICOMPQ = 0. If ICOMPQ = 1 and the I-th subproblem is not square, on exit, S(I) contains the S-value of a Givens rotation related to the right null space of the I-th subproblem.

WORK (workspace) DOUBLE PRECISION array, dimension
(6 * N + (SMLSIZ + 1)*(SMLSIZ + 1)).

IWORK (workspace) INTEGER array.
Dimension must be at least (7 * N).

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = -i, the i-th argument had an illegal value.
> 0: if INFO = 1, a singular value did not converge

Further Details

=====

Based on contributions by

Ming Gu and Huan Ren, Computer Science Division, University of
California at Berkeley, USA

— dlasda.f —

```

SUBROUTINE DLASDA( ICOMPQ, SMLSIZ, N, SQRE, D, E, U, LDU, VT, K,
$                DIFL, DIFR, Z, POLES, GIVPTR, GIVCOL, LDGCOL,
$                PERM, GIVNUM, C, S, WORK, IWORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
INTEGER          ICOMPQ, INFO, LDGCOL, LDU, N, SMLSIZ, SQRE
*
*  .. Array Arguments ..
INTEGER          GIVCOL( LDGCOL, * ), GIVPTR( * ), IWORK( * ),
$                K( * ), PERM( LDGCOL, * )
DOUBLE PRECISION C( * ), D( * ), DIFL( LDU, * ), DIFR( LDU, * ),
$                E( * ), GIVNUM( LDU, * ), POLES( LDU, * ),
$                S( * ), U( LDU, * ), VT( LDU, * ), WORK( * ),
$                Z( LDU, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
INTEGER          I, I1, IC, IDXQ, IDXQI, IM1, INODE, ITEMP, IWK,
$                J, LF, LL, LVL, LVL2, M, NCC, ND, NDB1, NDIML,
$                NDIMR, NL, NLF, NLP1, NLVL, NR, NRF, NRP1, NRU,
$                NWORK1, NWORK2, SMLSZP, SQREI, VF, VFI, VL, VLI
DOUBLE PRECISION ALPHA, BETA
*
*  ..
*  .. External Subroutines ..
EXTERNAL          DCOPY, DLASD6, DLASDQ, DLASDT, DLASET, XERBLA
*
*  ..
*  .. Executable Statements ..

```

```

*
*   Test the input parameters.
*
      INFO = 0
*
      IF( ( ICOMPQ.LT.0 ) .OR. ( ICOMPQ.GT.1 ) ) THEN
        INFO = -1
      ELSE IF( SMLSIZ.LT.3 ) THEN
        INFO = -2
      ELSE IF( N.LT.0 ) THEN
        INFO = -3
      ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN
        INFO = -4
      ELSE IF( LDU.LT.( N+SQRE ) ) THEN
        INFO = -8
      ELSE IF( LDGCOL.LT.N ) THEN
        INFO = -17
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DLASDA', -INFO )
        RETURN
      END IF
*
      M = N + SQRE
*
*   If the input matrix is too small, call DLASDQ to find the SVD.
*
      IF( N.LE.SMLSIZ ) THEN
        IF( ICOMPQ.EQ.0 ) THEN
          CALL DLASDQ( 'U', SQRE, N, 0, 0, 0, 0, D, E, VT, LDU, U, LDU,
$              U, LDU, WORK, INFO )
        ELSE
          CALL DLASDQ( 'U', SQRE, N, M, N, 0, 0, D, E, VT, LDU, U, LDU,
$              U, LDU, WORK, INFO )
        END IF
        RETURN
      END IF
*
*   Book-keeping and set up the computation tree.
*
      INODE = 1
      NDIML = INODE + N
      NDIMR = NDIML + N
      IDXQ = NDIMR + N
      IWK = IDXQ + N
*
      NCC = 0
      NRU = 0
*
      SMLSZP = SMLSIZ + 1

```

```

VF = 1
VL = VF + M
NWORK1 = VL + M
NWORK2 = NWORK1 + SMLSZP*SMLSZP
*
CALL DLASDT( N, NLVL, ND, IWORK( INODE ), IWORK( NDIML ),
$           IWORK( NDIMR ), SMLSIZ )
*
*   for the nodes on bottom level of the tree, solve
*   their subproblems by DLASDQ.
*
NDB1 = ( ND+1 ) / 2
DO 30 I = NDB1, ND
*
*   IC : center row of each node
*   NL : number of rows of left  subproblem
*   NR : number of rows of right subproblem
*   NLF: starting row of the left  subproblem
*   NRF: starting row of the right subproblem
*
I1 = I - 1
IC = IWORK( INODE+I1 )
NL = IWORK( NDIML+I1 )
NLP1 = NL + 1
NR = IWORK( NDIMR+I1 )
NLF = IC - NL
NRF = IC + 1
IDXQI = IDXQ + NLF - 2
VFI = VF + NLF - 1
VLI = VL + NLF - 1
SQREI = 1
IF( ICOMPQ.EQ.0 ) THEN
    CALL DLASET( 'A', NLP1, NLP1, ZERO, ONE, WORK( NWORK1 ),
$             SMLSZP )
    CALL DLASDQ( 'U', SQREI, NL, NLP1, NRU, NCC, D( NLF ),
$             E( NLF ), WORK( NWORK1 ), SMLSZP,
$             WORK( NWORK2 ), NL, WORK( NWORK2 ), NL,
$             WORK( NWORK2 ), INFO )
    ITEMP = NWORK1 + NL*SMLSZP
    CALL DCOPY( NLP1, WORK( NWORK1 ), 1, WORK( VFI ), 1 )
    CALL DCOPY( NLP1, WORK( ITEMP ), 1, WORK( VLI ), 1 )
ELSE
    CALL DLASET( 'A', NL, NL, ZERO, ONE, U( NLF, 1 ), LDU )
    CALL DLASET( 'A', NLP1, NLP1, ZERO, ONE, VT( NLF, 1 ), LDU )
    CALL DLASDQ( 'U', SQREI, NL, NLP1, NL, NCC, D( NLF ),
$             E( NLF ), VT( NLF, 1 ), LDU, U( NLF, 1 ), LDU,
$             U( NLF, 1 ), LDU, WORK( NWORK1 ), INFO )
    CALL DCOPY( NLP1, VT( NLF, 1 ), 1, WORK( VFI ), 1 )
    CALL DCOPY( NLP1, VT( NLF, NLP1 ), 1, WORK( VLI ), 1 )
END IF

```

```

      IF( INFO.NE.0 ) THEN
        RETURN
      END IF
      DO 10 J = 1, NL
        IWORK( IDXQI+J ) = J
10    CONTINUE
      IF( ( I.EQ.ND ) .AND. ( SQRE.EQ.0 ) ) THEN
        SQREI = 0
      ELSE
        SQREI = 1
      END IF
      IDXQI = IDXQI + NLP1
      VFI = VFI + NLP1
      VLI = VLI + NLP1
      NRP1 = NR + SQREI
      IF( ICOMPQ.EQ.0 ) THEN
        CALL DLASET( 'A', NRP1, NRP1, ZERO, ONE, WORK( NWORK1 ),
$          SMLSZP )
        CALL DLASDQ( 'U', SQREI, NR, NRP1, NRU, NCC, D( NRF ),
$          E( NRF ), WORK( NWORK1 ), SMLSZP,
$          WORK( NWORK2 ), NR, WORK( NWORK2 ), NR,
$          WORK( NWORK2 ), INFO )
        ITEMP = NWORK1 + ( NRP1-1 )*SMLSZP
        CALL DCOPY( NRP1, WORK( NWORK1 ), 1, WORK( VFI ), 1 )
        CALL DCOPY( NRP1, WORK( ITEMP ), 1, WORK( VLI ), 1 )
      ELSE
        CALL DLASET( 'A', NR, NR, ZERO, ONE, U( NRF, 1 ), LDU )
        CALL DLASET( 'A', NRP1, NRP1, ZERO, ONE, VT( NRF, 1 ), LDU )
        CALL DLASDQ( 'U', SQREI, NR, NRP1, NR, NCC, D( NRF ),
$          E( NRF ), VT( NRF, 1 ), LDU, U( NRF, 1 ), LDU,
$          U( NRF, 1 ), LDU, WORK( NWORK1 ), INFO )
        CALL DCOPY( NRP1, VT( NRF, 1 ), 1, WORK( VFI ), 1 )
        CALL DCOPY( NRP1, VT( NRF, NRP1 ), 1, WORK( VLI ), 1 )
      END IF
      IF( INFO.NE.0 ) THEN
        RETURN
      END IF
      DO 20 J = 1, NR
        IWORK( IDXQI+J ) = J
20    CONTINUE
30  CONTINUE
*
*    Now conquer each subproblem bottom-up.
*
      J = 2**NLVL
      DO 50 LVL = NLVL, 1, -1
        LVL2 = LVL*2 - 1
*
*    Find the first node LF and last node LL on
*    the current level LVL.

```

```

*
      IF( LVL.EQ.1 ) THEN
        LF = 1
        LL = 1
      ELSE
        LF = 2**( LVL-1 )
        LL = 2*LF - 1
      END IF
      DO 40 I = LF, LL
        IM1 = I - 1
        IC = IWORK( INODE+IM1 )
        NL = IWORK( NDI ML+IM1 )
        NR = IWORK( NDI MR+IM1 )
        NLF = IC - NL
        NRF = IC + 1
        IF( I.EQ.LL ) THEN
          SQREI = SQRE
        ELSE
          SQREI = 1
        END IF
        VFI = VF + NLF - 1
        VLI = VL + NLF - 1
        IDXQI = IDXQ + NLF - 1
        ALPHA = D( IC )
        BETA = E( IC )
        IF( ICOMPQ.EQ.0 ) THEN
          CALL DLASD6( ICOMPQ, NL, NR, SQREI, D( NLF ),
$              WORK( VFI ), WORK( VLI ), ALPHA, BETA,
$              IWORK( IDXQI ), PERM, GIVPTR( 1 ), GIVCOL,
$              LDGCOL, GIVNUM, LDU, POLES, DIFL, DIFR, Z,
$              K( 1 ), C( 1 ), S( 1 ), WORK( NWORK1 ),
$              IWORK( IWK ), INFO )
        ELSE
          J = J - 1
          CALL DLASD6( ICOMPQ, NL, NR, SQREI, D( NLF ),
$              WORK( VFI ), WORK( VLI ), ALPHA, BETA,
$              IWORK( IDXQI ), PERM( NLF, LVL ),
$              GIVPTR( J ), GIVCOL( NLF, LVL2 ), LDGCOL,
$              GIVNUM( NLF, LVL2 ), LDU,
$              POLES( NLF, LVL2 ), DIFL( NLF, LVL ),
$              DIFR( NLF, LVL2 ), Z( NLF, LVL ), K( J ),
$              C( J ), S( J ), WORK( NWORK1 ),
$              IWORK( IWK ), INFO )
        END IF
        IF( INFO.NE.0 ) THEN
          RETURN
        END IF
      40 CONTINUE
      50 CONTINUE
*

```



```

                                nrp1 nru nwork1 nwork2 smlszp sqrei
                                vf vfi vl vli))

(setf info 0)
(cond
  ((or (< icompq 0) (> icompq 1))
    (setf info -1))
  (< smlsiz 3)
    (setf info -2))
  (< n 0)
    (setf info -3))
  ((or (< sqre 0) (> sqre 1))
    (setf info -4))
  (< ldu (f2cl-lib:int-add n sqre))
    (setf info -8))
  (< ldgcol n)
    (setf info -17)))
(cond
  (/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DLASDA" (f2cl-lib:int-sub info))
    (go end_label)))
(setf m (f2cl-lib:int-add n sqre))
(cond
  (<= n smlsiz)
    (cond
      ((= icompq 0)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10 var-11 var-12 var-13 var-14 var-15)
          (dlasdq "U" sqre n 0 0 0 d e vt ldu u ldu u ldu work info)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                           var-7 var-8 var-9 var-10 var-11 var-12 var-13
                           var-14))
            (setf info var-15)))
        (t
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
             var-9 var-10 var-11 var-12 var-13 var-14 var-15)
            (dlasdq "U" sqre n m n 0 d e vt ldu u ldu u ldu work info)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                             var-7 var-8 var-9 var-10 var-11 var-12 var-13
                             var-14))
              (setf info var-15))))
        (go end_label)))
    (setf inode 1)
    (setf ndiml (f2cl-lib:int-add inode n))
    (setf ndimr (f2cl-lib:int-add ndiml n))
    (setf idxq (f2cl-lib:int-add ndimr n))
    (setf iwk (f2cl-lib:int-add idxq n))

```

```

(setf ncc 0)
(setf nru 0)
(setf smlszp (f2cl-lib:int-add smlsiz 1))
(setf vf 1)
(setf vl (f2cl-lib:int-add vf m))
(setf nwork1 (f2cl-lib:int-add vl m))
(setf nwork2
  (f2cl-lib:int-add nwork1 (f2cl-lib:int-mul smlszp smlszp)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (dlasdt n nlvl nd
    (f2cl-lib:array-slice iwork fixnum (inode) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (ndiml) ((1 *)))
    (f2cl-lib:array-slice iwork fixnum (ndimr) ((1 *)))
    smlsiz)
  (declare (ignore var-0 var-3 var-4 var-5 var-6))
  (setf nlvl var-1)
  (setf nd var-2))
(setf ndb1 (the fixnum (truncate (+ nd 1) 2)))
(f2cl-lib:fdo (i ndb1 (f2cl-lib:int-add i 1))
  (> i nd) nil)
(tagbody
  (setf i1 (f2cl-lib:int-sub i 1))
  (setf ic
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add inode i1))
      ((1 *))
      iwork-%offset%))
  (setf nl
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add ndiml i1))
      ((1 *))
      iwork-%offset%))
  (setf nlp1 (f2cl-lib:int-add nl 1))
  (setf nr
    (f2cl-lib:fref iwork-%data%
      ((f2cl-lib:int-add ndimr i1))
      ((1 *))
      iwork-%offset%))
  (setf nlf (f2cl-lib:int-sub ic nl))
  (setf nrf (f2cl-lib:int-add ic 1))
  (setf idxqi (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 2))
  (setf vfi (f2cl-lib:int-sub (f2cl-lib:int-add vf nlf) 1))
  (setf vli (f2cl-lib:int-sub (f2cl-lib:int-add vl nlf) 1))
  (setf sqrei 1)
  (cond
    ((= icompq 0)
     (dlaset "A" nlp1 nlp1 zero one
       (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
       smlszp)
     (multiple-value-bind

```



```

      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13 var-14 var-15)
(dlasdq "U" sqrei nl nlp1 nru ncc
 (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
 (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
 (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
 smlszp
 (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
 nl
 (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
 nl
 (f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
 info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                 var-7 var-8 var-9 var-10 var-11 var-12 var-13
                 var-14))
(setf info var-15))
(setf itemp
  (f2cl-lib:int-add nwork1 (f2cl-lib:int-mul nl smlszp)))
(dcopy nlp1
 (f2cl-lib:array-slice work double-float (nwork1) ((1 *))) 1
 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
(dcopy nlp1
 (f2cl-lib:array-slice work double-float (itemp) ((1 *))) 1
 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1))
(t
 (dlaset "A" nl nl zero one
  (f2cl-lib:array-slice u double-float (nlf 1) ((1 ldu) (1 *)))
  ldu)
 (dlaset "A" nlp1 nlp1 zero one
  (f2cl-lib:array-slice vt double-float (nlf 1) ((1 ldu) (1 *)))
  ldu)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11 var-12 var-13 var-14 var-15)
  (dlasdq "U" sqrei nl nlp1 nl ncc
   (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
   (f2cl-lib:array-slice e double-float (nlf) ((1 *)))
   (f2cl-lib:array-slice vt
    double-float
    (nlf 1)
    ((1 ldu) (1 *)))
   ldu
   (f2cl-lib:array-slice u
    double-float
    (nlf 1)
    ((1 ldu) (1 *)))
   ldu
   (f2cl-lib:array-slice u
    double-float

```

```

                                (nlf 1)
                                ((1 ldu) (1 *)))

    ldu
    (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
    info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7 var-8 var-9 var-10 var-11 var-12 var-13
                    var-14))
    (setf info var-15))
    (dcopy nlp1
    (f2cl-lib:array-slice vt double-float (nlf 1) ((1 ldu) (1 *)))
    1 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
    (dcopy nlp1
    (f2cl-lib:array-slice vt
                        double-float
                        (nlf nlp1)
                        ((1 ldu) (1 *)))
    1 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1)))
    (cond
      ((/= info 0)
       (go end_label)))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j nl) nil)
    (tagbody
      (setf (f2cl-lib:fref iwork-%data%
                        ((f2cl-lib:int-add idxqi j))
                        ((1 *))
                        iwork-%offset%)
            j)))
    (cond
      ((and (= i nd) (= sqre 0))
       (setf sqrei 0))
      (t
       (setf sqrei 1)))
    (setf idxqi (f2cl-lib:int-add idxqi nlp1))
    (setf vfi (f2cl-lib:int-add vfi nlp1))
    (setf vli (f2cl-lib:int-add vli nlp1))
    (setf nrp1 (f2cl-lib:int-add nr sqrei))
    (cond
      ((= icompq 0)
       (dlaset "A" nrp1 nrp1 zero one
        (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
        smlszp)
       (multiple-value-bind
         (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
          var-9 var-10 var-11 var-12 var-13 var-14 var-15)
         (dlasdq "U" sqrei nr nrp1 nru ncc
          (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
          (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
          (f2cl-lib:array-slice work double-float (nwork1) ((1 *))))

```

```

smlszp
(f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
nr
(f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
nr
(f2cl-lib:array-slice work double-float (nwork2) ((1 *)))
info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
               var-7 var-8 var-9 var-10 var-11 var-12 var-13
               var-14))
(setf info var-15))
(setf itemp
  (f2cl-lib:int-add nwork1
    (f2cl-lib:int-mul
      (f2cl-lib:int-sub nrp1 1)
      smlszp)))
(dcopy nrp1
  (f2cl-lib:array-slice work double-float (nwork1) ((1 *))) 1
  (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
(dcopy nrp1
  (f2cl-lib:array-slice work double-float (itemp) ((1 *))) 1
  (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1))
(t
  (dlaset "A" nr nr zero one
    (f2cl-lib:array-slice u double-float (nrf 1) ((1 ldu) (1 *)))
    ldu)
  (dlaset "A" nrp1 nrp1 zero one
    (f2cl-lib:array-slice vt double-float (nrf 1) ((1 ldu) (1 *)))
    ldu)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12 var-13 var-14 var-15)
    (dlasdq "U" sqrei nr nrp1 nr ncc
      (f2cl-lib:array-slice d double-float (nrf) ((1 *)))
      (f2cl-lib:array-slice e double-float (nrf) ((1 *)))
      (f2cl-lib:array-slice vt
        double-float
        (nrf 1)
        ((1 ldu) (1 *)))

      ldu
      (f2cl-lib:array-slice u
        double-float
        (nrf 1)
        ((1 ldu) (1 *)))

      ldu
      (f2cl-lib:array-slice u
        double-float
        (nrf 1)
        ((1 ldu) (1 *)))

      ldu

```

```

        (f2cl-lib:array-slice work double-float (nwork1) ((1 *)))
        info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7 var-8 var-9 var-10 var-11 var-12 var-13
                      var-14))
      (setf info var-15))
    (dcopy nrp1
      (f2cl-lib:array-slice vt double-float (nrf 1) ((1 ldu) (1 *)))
      1 (f2cl-lib:array-slice work double-float (vfi) ((1 *))) 1)
    (dcopy nrp1
      (f2cl-lib:array-slice vt
                            double-float
                            (nrf nrp1)
                            ((1 ldu) (1 *)))
      1 (f2cl-lib:array-slice work double-float (vli) ((1 *))) 1)))
  (cond
    ((/= info 0)
     (go end_label)))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j nr) nil)
  (tagbody
    (setf (f2cl-lib:fref iwork-%data%
                        ((f2cl-lib:int-add idxqi j))
                        ((1 *))
                        iwork-%offset%)
          j))))
  (setf j (expt 2 nlvl))
  (f2cl-lib:fdo (lvl nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1)))
    (> lvl 1) nil)
  (tagbody
    (setf lvl2 (f2cl-lib:int-sub (f2cl-lib:int-mul lvl 2) 1))
    (cond
      ((= lvl 1)
       (setf lf 1)
       (setf ll 1))
      (t
       (setf lf (expt 2 (f2cl-lib:int-sub lvl 1)))
       (setf ll (f2cl-lib:int-sub (f2cl-lib:int-mul 2 lf) 1))))
    (f2cl-lib:fdo (i lf (f2cl-lib:int-add i 1))
      (> i ll) nil)
    (tagbody
      (setf im1 (f2cl-lib:int-sub i 1))
      (setf ic
        (f2cl-lib:fref iwork-%data%
                      ((f2cl-lib:int-add inode im1))
                      ((1 *))
                      iwork-%offset%))
      (setf nl
        (f2cl-lib:fref iwork-%data%
                      ((f2cl-lib:int-add ndim1 im1))

```

```

                                ((1 *))
                                iwork-%offset%)
(setf nr
  (f2cl-lib:fref iwork-%data%
    ((f2cl-lib:int-add ndimr im1))
    ((1 *))
    iwork-%offset%))
(setf nlf (f2cl-lib:int-sub ic n1))
(setf nrf (f2cl-lib:int-add ic 1))
(cond
  ((= i 11)
    (setf sqrei sqre))
  (t
    (setf sqrei 1)))
(setf vfi (f2cl-lib:int-sub (f2cl-lib:int-add vf nlf) 1))
(setf vli (f2cl-lib:int-sub (f2cl-lib:int-add vl nlf) 1))
(setf idxqi (f2cl-lib:int-sub (f2cl-lib:int-add idxq nlf) 1))
(setf alpha (f2cl-lib:fref d-%data% (ic) ((1 *)) d-%offset%))
(setf beta (f2cl-lib:fref e-%data% (ic) ((1 *)) e-%offset%))
(cond
  ((= icompq 0)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11 var-12 var-13 var-14 var-15
       var-16 var-17 var-18 var-19 var-20 var-21 var-22
       var-23 var-24 var-25)
      (dlasd6 icompq n1 nr sqrei
        (f2cl-lib:array-slice d double-float (nlf) ((1 *)))
        (f2cl-lib:array-slice work double-float (vfi) ((1 *)))
        (f2cl-lib:array-slice work double-float (vli) ((1 *)))
        alpha beta
        (f2cl-lib:array-slice iwork
          fixnum
          (idxqi)
          ((1 *)))

        perm
        (f2cl-lib:fref givptr-%data%
          (1)
          ((1 *))
          givptr-%offset%)
        givcol ldgcol givnum ldu poles difl difr z
        (f2cl-lib:fref k-%data% (1) ((1 *)) k-%offset%)
        (f2cl-lib:fref c-%data% (1) ((1 *)) c-%offset%)
        (f2cl-lib:fref s-%data% (1) ((1 *)) s-%offset%)
        (f2cl-lib:array-slice work
          double-float
          (nwork1)
          ((1 *)))

        (f2cl-lib:array-slice iwork
          fixnum

```



```

(f2cl-lib:array-slice givnum
  double-float
  (nlf lv12)
  ((1 ldu) (1 *)))
ldu
(f2cl-lib:array-slice poles
  double-float
  (nlf lv12)
  ((1 ldu) (1 *)))
(f2cl-lib:array-slice difl
  double-float
  (nlf lv1)
  ((1 ldu) (1 *)))
(f2cl-lib:array-slice difr
  double-float
  (nlf lv12)
  ((1 ldu) (1 *)))
(f2cl-lib:array-slice z
  double-float
  (nlf lv1)
  ((1 ldu) (1 *)))
(f2cl-lib:fref k-%data% (j) ((1 *)) k-%offset%)
(f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%)
(f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%)
(f2cl-lib:array-slice work
  double-float
  (nwork1)
  ((1 *)))
(f2cl-lib:array-slice iwork
  fixnum
  (iwk)
  ((1 *)))

info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
  var-9 var-10 var-12 var-13 var-14 var-15
  var-16 var-17 var-18 var-19 var-23
  var-24))
(setf alpha var-7)
(setf beta var-8)
(setf (f2cl-lib:fref givptr-%data%
  (j)
  ((1 *))
  givptr-%offset%)
  var-11)
(setf (f2cl-lib:fref k-%data% (j) ((1 *)) k-%offset%)
  var-20)
(setf (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%)
  var-21)
(setf (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%)
  var-22)

```

dlasdq LAPACK

— dlasdq.input —

```
)set break resume
)sys rm -f dlasdq.output
)spool dlasdq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlasdq.help —

=====

dlasdq examples

=====

=====

Man Page Details

=====

NAME

DLASDQ - the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired

SYNOPSIS

SUBROUTINE DLASDQ(UPLO, SQRE, N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C, LDC, WORK, INFO)

CHARACTER UPLO

INTEGER INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU, SQRE

DOUBLE PRECISION C(LDC, *), D(*), E(*), U(LDU, *), VT(LDVT, *), WORK(*)

Purpose

=====

DLASDQ computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired. Letting B denote the input bidiagonal matrix, the algorithm computes orthogonal matrices Q and P such that $B = Q * S * P'$ (P' denotes the transpose of P). The singular values S are overwritten on D.

The input matrix U is changed to $U * Q$ if desired.

The input matrix VT is changed to $P' * VT$ if desired.

The input matrix C is changed to $Q' * C$ if desired.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3, for a detailed description of the algorithm.

Arguments

=====

UPLO (input) CHARACTER*1
 On entry, UPLO specifies whether the input bidiagonal matrix is upper or lower bidiagonal, and whether it is square or not.
 UPLO = 'U' or 'u' B is upper bidiagonal.
 UPLO = 'L' or 'l' B is lower bidiagonal.

SQRE (input) INTEGER
 = 0: then the input matrix is N-by-N.
 = 1: then the input matrix is N-by-(N+1) if UPLU = 'U' and (N+1)-by-N if UPLU = 'L'.

 The bidiagonal matrix has
 N = NL + NR + 1 rows and
 M = N + SQRE \geq N columns.

N (input) INTEGER
 On entry, N specifies the number of rows and columns in the matrix. N must be at least 0.

NCVT (input) INTEGER
 On entry, NCVT specifies the number of columns of the matrix VT. NCVT must be at least 0.

NRU (input) INTEGER
 On entry, NRU specifies the number of rows of the matrix U. NRU must be at least 0.

NCC (input) INTEGER
 On entry, NCC specifies the number of columns of the matrix C. NCC must be at least 0.

D (input/output) DOUBLE PRECISION array, dimension (N)
 On entry, D contains the diagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, D contains the singular values in ascending order.

E (input/output) DOUBLE PRECISION array.
 dimension is (N-1) if SQRE = 0 and N if SQRE = 1.
 On entry, the entries of E contain the offdiagonal entries of the bidiagonal matrix whose SVD is desired. On normal exit, E will contain 0. If the algorithm does not converge, D and E will contain the diagonal and superdiagonal entries of a bidiagonal matrix orthogonally equivalent to the one given as input.

VT (input/output) DOUBLE PRECISION array, dimension (LDVT, NCVT)
 On entry, contains a matrix which on exit has been premultiplied by P', dimension N-by-NCVT if SQRE = 0 and (N+1)-by-NCVT if SQRE = 1 (not referenced if NCVT=0).

LDVT (input) INTEGER
 On entry, LDVT specifies the leading dimension of VT as declared in the calling (sub) program. LDVT must be at least 1. If NCVT is nonzero LDVT must also be at least N.

U (input/output) DOUBLE PRECISION array, dimension (LDU, N)
 On entry, contains a matrix which on exit has been postmultiplied by Q, dimension NRU-by-N if SQRE = 0 and NRU-by-(N+1) if SQRE = 1 (not referenced if NRU=0).

LDU (input) INTEGER
 On entry, LDU specifies the leading dimension of U as declared in the calling (sub) program. LDU must be at least $\max(1, \text{NRU})$.

C (input/output) DOUBLE PRECISION array, dimension (LDC, NCC)
 On entry, contains an N-by-NCC matrix which on exit has been premultiplied by Q' dimension N-by-NCC if SQRE = 0 and (N+1)-by-NCC if SQRE = 1 (not referenced if NCC=0).

LDC (input) INTEGER
 On entry, LDC specifies the leading dimension of C as declared in the calling (sub) program. LDC must be at least 1. If NCC is nonzero, LDC must also be at least N.

WORK (workspace) DOUBLE PRECISION array, dimension (4*N)
 Workspace. Only referenced if one of NCVT, NRU, or NCC is nonzero, and if N is at least 2.

INFO (output) INTEGER
 On exit, a value of 0 indicates a successful exit.
 If INFO < 0, argument number -INFO is illegal.
 If INFO > 0, the algorithm did not converge, and INFO specifies how many superdiagonals did not converge.

Further Details
 =====

Based on contributions by
 Ming Gu and Huan Ren, Computer Science Division, University of
 California at Berkeley, USA

— dlasdq.f —

SUBROUTINE DLASDQ(UPLO, SQRE, N, NCVT, NRU, NCC, D, E, VT, LDVT,

```

$          U, LDU, C, LDC, WORK, INFO )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1999
*
*    .. Scalar Arguments ..
*    CHARACTER          UPLO
*    INTEGER            INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU, SQRE
*
*    ..
*    .. Array Arguments ..
*    DOUBLE PRECISION   C( LDC, * ), D( * ), E( * ), U( LDU, * ),
$                      VT( LDVT, * ), WORK( * )
*
*    ..
*
*    =====
*
*    .. Parameters ..
*    DOUBLE PRECISION   ZERO
*    PARAMETER          ( ZERO = 0.0D+0 )
*
*    ..
*    .. Local Scalars ..
*    LOGICAL            ROTATE
*    INTEGER            I, ISUB, IUPLO, J, NP1, SQRE1
*    DOUBLE PRECISION   CS, R, SMIN, SN
*
*    ..
*    .. External Subroutines ..
*    EXTERNAL           DBDSQR, DLARTG, DLASR, DSWAP, XERBLA
*
*    ..
*    .. External Functions ..
*    LOGICAL            LSAME
*    EXTERNAL           LSAME
*
*    ..
*    .. Intrinsic Functions ..
*    INTRINSIC          MAX
*
*    ..
*    .. Executable Statements ..
*
*    Test the input parameters.
*
*
*    INFO = 0
*    IUPLO = 0
*    IF( LSAME( UPLO, 'U' ) )
$      IUPLO = 1
*    IF( LSAME( UPLO, 'L' ) )
$      IUPLO = 2
*    IF( IUPLO.EQ.0 ) THEN
*      INFO = -1
*    ELSE IF( ( SQRE.LT.0 ) .OR. ( SQRE.GT.1 ) ) THEN

```

```

        INFO = -2
      ELSE IF( N.LT.0 ) THEN
        INFO = -3
      ELSE IF( NCVT.LT.0 ) THEN
        INFO = -4
      ELSE IF( NRU.LT.0 ) THEN
        INFO = -5
      ELSE IF( NCC.LT.0 ) THEN
        INFO = -6
      ELSE IF( ( NCVT.EQ.0 .AND. LDVT.LT.1 ) .OR.
$      ( NCVT.GT.0 .AND. LDVT.LT.MAX( 1, N ) ) ) THEN
        INFO = -10
      ELSE IF( LDU.LT.MAX( 1, NRU ) ) THEN
        INFO = -12
      ELSE IF( ( NCC.EQ.0 .AND. LDC.LT.1 ) .OR.
$      ( NCC.GT.0 .AND. LDC.LT.MAX( 1, N ) ) ) THEN
        INFO = -14
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DLASDQ', -INFO )
        RETURN
      END IF
      IF( N.EQ.0 )
$      RETURN
*
*      ROTATE is true if any singular vectors desired, false otherwise
*
      ROTATE = ( NCVT.GT.0 ) .OR. ( NRU.GT.0 ) .OR. ( NCC.GT.0 )
      NP1 = N + 1
      SQRE1 = SQRE
*
*      If matrix non-square upper bidiagonal, rotate to be lower
*      bidiagonal. The rotations are on the right.
*
      IF( ( IUPLO.EQ.1 ) .AND. ( SQRE1.EQ.1 ) ) THEN
        DO 10 I = 1, N - 1
          CALL DLARTG( D( I ), E( I ), CS, SN, R )
          D( I ) = R
          E( I ) = SN*D( I+1 )
          D( I+1 ) = CS*D( I+1 )
          IF( ROTATE ) THEN
            WORK( I ) = CS
            WORK( N+I ) = SN
          END IF
10      CONTINUE
        CALL DLARTG( D( N ), E( N ), CS, SN, R )
        D( N ) = R
        E( N ) = ZERO
        IF( ROTATE ) THEN
          WORK( N ) = CS

```

```

        WORK( N+N ) = SN
    END IF
    IUPLO = 2
    SQRE1 = 0
*
*   Update singular vectors if desired.
*
    IF( NCVT.GT.0 )
$       CALL DLASR( 'L', 'V', 'F', NP1, NCVT, WORK( 1 ),
$               WORK( NP1 ), VT, LDVT )
    END IF
*
*   If matrix lower bidiagonal, rotate to be upper bidiagonal
*   by applying Givens rotations on the left.
*
    IF( IUPLO.EQ.2 ) THEN
        DO 20 I = 1, N - 1
            CALL DLARTG( D( I ), E( I ), CS, SN, R )
            D( I ) = R
            E( I ) = SN*D( I+1 )
            D( I+1 ) = CS*D( I+1 )
            IF( ROTATE ) THEN
                WORK( I ) = CS
                WORK( N+I ) = SN
            END IF
20      CONTINUE
*
*   If matrix (N+1)-by-N lower bidiagonal, one additional
*   rotation is needed.
*
    IF( SQRE1.EQ.1 ) THEN
        CALL DLARTG( D( N ), E( N ), CS, SN, R )
        D( N ) = R
        IF( ROTATE ) THEN
            WORK( N ) = CS
            WORK( N+N ) = SN
        END IF
    END IF
*
*   Update singular vectors if desired.
*
    IF( NRU.GT.0 ) THEN
        IF( SQRE1.EQ.0 ) THEN
            CALL DLASR( 'R', 'V', 'F', NRU, N, WORK( 1 ),
$                   WORK( NP1 ), U, LDU )
        ELSE
            CALL DLASR( 'R', 'V', 'F', NRU, NP1, WORK( 1 ),
$                   WORK( NP1 ), U, LDU )
        END IF
    END IF

```

```

      IF( NCC.GT.0 ) THEN
        IF( SQRE1.EQ.0 ) THEN
          CALL DLASR( 'L', 'V', 'F', N, NCC, WORK( 1 ),
$              WORK( NP1 ), C, LDC )
        ELSE
          CALL DLASR( 'L', 'V', 'F', NP1, NCC, WORK( 1 ),
$              WORK( NP1 ), C, LDC )
        END IF
      END IF
END IF

*
* Call DBDSQR to compute the SVD of the reduced real
* N-by-N upper bidiagonal matrix.
*
CALL DBDSQR( 'U', N, NCVT, NRU, NCC, D, E, VT, LDVT, U, LDU, C,
$          LDC, WORK, INFO )
*
* Sort the singular values into ascending order (insertion sort on
* singular values, but only one transposition per singular vector)
*
DO 40 I = 1, N
*
*   Scan for smallest D(I).
*
  ISUB = I
  SMIN = D( I )
  DO 30 J = I + 1, N
    IF( D( J ).LT.SMIN ) THEN
      ISUB = J
      SMIN = D( J )
    END IF
  30 CONTINUE
  IF( ISUB.NE.I ) THEN
*
*   Swap singular values and vectors.
*
    D( ISUB ) = D( I )
    D( I ) = SMIN
    IF( NCVT.GT.0 )
$      CALL DSWAP( NCVT, VT( ISUB, 1 ), LDVT, VT( I, 1 ), LDVT )
    IF( NRU.GT.0 )
$      CALL DSWAP( NRU, U( 1, ISUB ), 1, U( 1, I ), 1 )
    IF( NCC.GT.0 )
$      CALL DSWAP( NCC, C( ISUB, 1 ), LDC, C( I, 1 ), LDC )
  END IF
40 CONTINUE

*
RETURN
*
* End of DLASDQ

```

*
END

— LAPACK dlasdq —

```
(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasdq (uplo sqre n ncvr nru ncc d e vt ldvt u ldu c ldc work info)
    (declare (type (simple-array double-float (*)) work c u vt e d)
      (type fixnum info ldc ldu ldvt ncc nru ncvr n sqre)
      (type character uplo))
    (f2cl-lib:with-multi-array-data
      ((uplo character uplo-%data% uplo-%offset%)
       (d double-float d-%data% d-%offset%)
       (e double-float e-%data% e-%offset%)
       (vt double-float vt-%data% vt-%offset%)
       (u double-float u-%data% u-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((cs 0.0) (r 0.0) (smin 0.0) (sn 0.0) (i 0) (isub 0) (iuplo 0)
        (j 0) (np1 0) (sqre1 0) (rotate nil))
        (declare (type (double-float) cs r smin sn)
          (type fixnum i isub iuplo j np1 sqre1)
          (type (member t nil) rotate))
        (setf info 0)
        (setf iuplo 0)
        (if (char-equal uplo #\U) (setf iuplo 1))
        (if (char-equal uplo #\L) (setf iuplo 2))
        (cond
          ((= iuplo 0)
            (setf info -1))
          ((or (< sqre 0) (> sqre 1))
            (setf info -2))
          ((< n 0)
            (setf info -3))
          ((< ncvr 0)
            (setf info -4))
          ((< nru 0)
            (setf info -5))
          ((< ncc 0)
            (setf info -6))
          ((or (and (= ncvr 0) (< ldvt 1))
              (and (> ncvr 0)
                (< ldvt
                  (max (the fixnum 1)
                       (the fixnum n)))))))
```



```

      (setf info -10))
      (< ldu (max (the fixnum 1) (the fixnum nru)))
      (setf info -12))
      ((or (and (= ncc 0) (< ldc 1))
           (and (> ncc 0)
                (< ldc
                 (max (the fixnum 1)
                      (the fixnum n))))))
      (setf info -14)))
      (cond
       ((/= info 0)
        (error
         " ** On entry to ~a parameter number ~a had an illegal value~%"
         "DLASDQ" (f2cl-lib:int-sub info))
        (go end_label)))
      (if (= n 0) (go end_label))
      (setf rotate (or (> ncvr 0) (> nru 0) (> ncc 0)))
      (setf np1 (f2cl-lib:int-add n 1))
      (setf sqre1 sqre)
      (cond
       ((and (= iuplo 1) (= sqre1 1))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                      ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
          (tagbody
           (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
             (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                     (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
             (declare (ignore var-0 var-1))
             (setf cs var-2)
             (setf sn var-3)
             (setf r var-4))
           (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
           (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
                 (* sn
                  (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-add i 1))
                                ((1 *))
                                d-%offset%)))
           (setf (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-add i 1))
                                ((1 *))
                                d-%offset%)
                 (* cs
                  (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-add i 1))
                                ((1 *))
                                d-%offset%)))
           (cond
            (rotate
             (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)

```

```

        cs)
        (setf (f2cl-lib:fref work-%data%
                           ((f2cl-lib:int-add n i))
                           ((1 *))
                           work-%offset%)
              sn))))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (dlartg (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
    (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) cs sn r)
  (declare (ignore var-0 var-1))
  (setf cs var-2)
  (setf sn var-3)
  (setf r var-4))
(setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) r)
(setf (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) zero)
(cond
  (rotate
    (setf (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%) cs)
    (setf (f2cl-lib:fref work-%data%
                           ((f2cl-lib:int-add n n))
                           ((1 *))
                           work-%offset%)
          sn)))
(setf iuplo 2)
(setf sqre1 0)
(if (> ncv1 0)
  (dlasr "L" "V" "F" np1 ncv1
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (np1) ((1 *))) vt
    ldvt))))
(cond
  (= iuplo 2)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlartg (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%) cs sn r)
      (declare (ignore var-0 var-1))
      (setf cs var-2)
      (setf sn var-3)
      (setf r var-4))
    (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) r)
    (setf (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%)
      (* sn
        (f2cl-lib:fref d-%data%
                       ((f2cl-lib:int-add i 1))
                       ((1 *))
                       d-%offset%)))
    (setf (f2cl-lib:fref d-%data%

```

```

((f2cl-lib:int-add i 1))
((1 *))
d-%offset%)
(* cs
  (f2cl-lib:fref d-%data%
    ((f2cl-lib:int-add i 1))
    ((1 *))
    d-%offset%)))
(cond
  (rotate
    (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
      cs)
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-add n i))
      ((1 *))
      work-%offset%)
      sn))))))
(cond
  ((= sqre1 1)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (dlartg (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
        (f2cl-lib:fref e-%data% (n) ((1 *)) e-%offset%) cs sn r)
      (declare (ignore var-0 var-1))
      (setf cs var-2)
      (setf sn var-3)
      (setf r var-4))
    (setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%) r)
    (cond
      (rotate
        (setf (f2cl-lib:fref work-%data% (n) ((1 *)) work-%offset%) cs)
        (setf (f2cl-lib:fref work-%data%
          ((f2cl-lib:int-add n n))
          ((1 *))
          work-%offset%)
          sn))))))
  (cond
    (> nru 0)
    (cond
      ((= sqre1 0)
        (dlasr "R" "V" "F" nru n
          (f2cl-lib:array-slice work double-float (1) ((1 *)))
          (f2cl-lib:array-slice work double-float (np1) ((1 *))) u ldu))
      (t
        (dlasr "R" "V" "F" nru np1
          (f2cl-lib:array-slice work double-float (1) ((1 *)))
          (f2cl-lib:array-slice work double-float (np1) ((1 *))) u
          ldu))))))
  (cond
    (> ncc 0)
    (cond

```

```

(= sqre1 0)
(dlasr "L" "V" "F" n ncc
  (f2cl-lib:array-slice work double-float (1) ((1 *)))
  (f2cl-lib:array-slice work double-float (np1) ((1 *))) c ldc))
(t
  (dlasr "L" "V" "F" np1 ncc
    (f2cl-lib:array-slice work double-float (1) ((1 *)))
    (f2cl-lib:array-slice work double-float (np1) ((1 *))) c
    ldc))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11 var-12 var-13 var-14)
  (dbdsqr "U" n ncv t nru ncc d e vt ldvt u ldu c ldc work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10 var-11 var-12 var-13))
  (setf info var-14))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
(tagbody
  (setf isub i)
  (setf smin (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (f2cl-lib:fdo (j (f2cl-lib:int-add i 1) (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (cond
      ((< (f2cl-lib:fref d (j) ((1 *))) smin)
        (setf isub j)
        (setf smin
          (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))))))
  (cond
    ((/= isub i)
      (setf (f2cl-lib:fref d-%data% (isub) ((1 *)) d-%offset%)
        (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
      (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) smin)
      (if (> ncv t 0)
        (dswap ncv t
          (f2cl-lib:array-slice vt
            double-float
            (isub 1)
            ((1 ldvt) (1 *)))
          ldvt
          (f2cl-lib:array-slice vt
            double-float
            (i 1)
            ((1 ldvt) (1 *)))
          ldvt))
        (if (> nru 0)
          (dswap nru
            (f2cl-lib:array-slice u
              double-float

```

```

                                (1 isub)
                                ((1 ldu) (1 *)))

1
(f2cl-lib:array-slice u double-float (1 i) ((1 ldu) (1 *)))
1))
(if (> ncc 0)
  (dswap ncc
    (f2cl-lib:array-slice c
                          double-float
                          (isub 1)
                          ((1 ldc) (1 *)))

    ldc
    (f2cl-lib:array-slice c double-float (i 1) ((1 ldc) (1 *)))
    ldc))))))
end_label
(return
  (values nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    nil
    info))))))

```

dlasdt LAPACK

— dlasdt.input —

```

)set break resume
)sys rm -f dlasdt.output
)spool dlasdt.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— dlasdt.help —

```
=====
dlasdt examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASDT - a tree of subproblems for bidiagonal divide and conquer

SYNOPSIS

```
SUBROUTINE DLASDT( N, LVL, ND, INODE, NDI ML, NDI MR, MSUB )
```

```
INTEGER          LVL, MSUB, N, ND
```

```
INTEGER          INODE( * ), NDI ML( * ), NDI MR( * )
```

Purpose

```
=====
```

DLASDT creates a tree of subproblems for bidiagonal divide and conquer.

Arguments

```
=====
```

N (input) INTEGER
On entry, the number of diagonal elements of the bidiagonal matrix.

LVL (output) INTEGER
On exit, the number of levels on the computation tree.

ND (output) INTEGER
On exit, the number of nodes on the tree.

INODE (output) INTEGER array, dimension (N)
On exit, centers of subproblems.

NDI ML (output) INTEGER array, dimension (N)
On exit, row dimensions of left children.

NDIMR (output) INTEGER array, dimension (N)
On exit, row dimensions of right children.

MSUB (input) INTEGER.
On entry, the maximum row dimension each subproblem at the
bottom of the tree can be of.

Further Details
=====

Based on contributions by
Ming Gu and Huan Ren, Computer Science Division, University of
California at Berkeley, USA

— dlasdt.f —

```

SUBROUTINE DLASDT( N, LVL, ND, INODE, NDIML, NDIMR, MSUB )
*
* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   June 30, 1999
*
*   .. Scalar Arguments ..
*   INTEGER          LVL, MSUB, N, ND
*
*   ..
*   .. Array Arguments ..
*   INTEGER          INODE( * ), NDIML( * ), NDIMR( * )
*
*   ..
*
* =====
*
*   .. Parameters ..
*   DOUBLE PRECISION  TWO
*   PARAMETER         ( TWO = 2.0D+0 )
*
*   ..
*   .. Local Scalars ..
*   INTEGER           I, IL, IR, LLST, MAXN, NCRNT, NLVL
*   DOUBLE PRECISION  TEMP
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC         DBLE, INT, LOG, MAX
*
*   ..
*   .. Executable Statements ..
*

```

```

*      Find the number of levels on the tree.
*
      MAXN = MAX( 1, N )
      TEMP = LOG( DBLE( MAXN ) / DBLE( MSUB+1 ) ) / LOG( TWO )
      LVL = INT( TEMP ) + 1
*
      I = N / 2
      INODE( 1 ) = I + 1
      NDIML( 1 ) = I
      NDIMR( 1 ) = N - I - 1
      IL = 0
      IR = 1
      LLST = 1
      DO 20 NLVL = 1, LVL - 1
*
*      Constructing the tree at (NLVL+1)-st level. The number of
*      nodes created on this level is LLST * 2.
*
      DO 10 I = 0, LLST - 1
          IL = IL + 2
          IR = IR + 2
          NCRNT = LLST + I
          NDIML( IL ) = NDIML( NCRNT ) / 2
          NDIMR( IL ) = NDIML( NCRNT ) - NDIML( IL ) - 1
          INODE( IL ) = INODE( NCRNT ) - NDIMR( IL ) - 1
          NDIML( IR ) = NDIMR( NCRNT ) / 2
          NDIMR( IR ) = NDIMR( NCRNT ) - NDIML( IR ) - 1
          INODE( IR ) = INODE( NCRNT ) + NDIML( IR ) + 1
10      CONTINUE
          LLST = LLST*2
20      CONTINUE
          ND = LLST*2 - 1
*
      RETURN
*
*      End of DLASDT
*
      END

```

— LAPACK dlasdt —

```

(let* ((two 2.0))
  (declare (type (double-float 2.0 2.0) two))
  (defun dlasdt (n lvl nd inode ndiml ndimr msub)
    (declare (type (simple-array fixnum (*)) ndimr ndiml inode)
              (type fixnum msub nd lvl n))

```



```

(f2cl-lib:with-multi-array-data
  ((inode fixnum inode-%data% inode-%offset%)
   (ndiml fixnum ndiml-%data% ndiml-%offset%)
   (ndimr fixnum ndimr-%data% ndimr-%offset%))
  (prog ((temp 0.0) (i 0) (il 0) (ir 0) (llst 0) (maxn 0) (ncrnt 0)
        (nlvl 0))
    (declare (type (double-float) temp)
              (type fixnum i il ir llst maxn ncrnt nlvl))
    (setf maxn (max (the fixnum 1) (the fixnum n)))
    (setf temp
      (/
        (f2cl-lib:flog
          (/ (coerce (realpart maxn) 'double-float)
             (coerce (realpart (f2cl-lib:int-add msub 1)) 'double-float)))
        (f2cl-lib:flog two)))
    (setf lvl (f2cl-lib:int-add (f2cl-lib:int temp) 1))
    (setf i (the fixnum (truncate n 2)))
    (setf (f2cl-lib:fref inode-%data% (1) ((1 *)) inode-%offset%)
          (f2cl-lib:int-add i 1))
    (setf (f2cl-lib:fref ndiml-%data% (1) ((1 *)) ndiml-%offset%) i)
    (setf (f2cl-lib:fref ndimr-%data% (1) ((1 *)) ndimr-%offset%)
          (f2cl-lib:int-sub n i 1))
    (setf il 0)
    (setf ir 1)
    (setf llst 1)
    (f2cl-lib:fdo (nlvl 1 (f2cl-lib:int-add nlvl 1))
      ((> nlvl (f2cl-lib:int-add lvl (f2cl-lib:int-sub 1))) nil)
      (tagbody
        (f2cl-lib:fdo (i 0 (f2cl-lib:int-add i 1))
          ((> i (f2cl-lib:int-add llst (f2cl-lib:int-sub 1)))
           nil)
          (tagbody
            (setf il (f2cl-lib:int-add il 2))
            (setf ir (f2cl-lib:int-add ir 2))
            (setf ncrnt (f2cl-lib:int-add llst i))
            (setf (f2cl-lib:fref ndiml-%data% (il) ((1 *)) ndiml-%offset%)
                  (the fixnum
                     (truncate
                      (f2cl-lib:fref ndiml-%data%
                                       (ncrnt)
                                       ((1 *))
                                       ndiml-%offset%)
                      2)))
            (setf (f2cl-lib:fref ndimr-%data% (il) ((1 *)) ndimr-%offset%)
                  (f2cl-lib:int-sub
                   (f2cl-lib:fref ndiml-%data%
                                   (ncrnt)
                                   ((1 *))
                                   ndiml-%offset%)
                   (f2cl-lib:fref ndiml-%data%
                                   (ncrnt)
                                   ((1 *))
                                   ndiml-%offset%)
                   (f2cl-lib:fref ndiml-%data%
                                   (ncrnt)
                                   ((1 *))
                                   ndiml-%offset%))

```

```

                                (il)
                                ((1 *))
                                ndiml-%offset%)
1))
(setf (f2cl-lib:fref inode-%data% (il) ((1 *)) inode-%offset%)
      (f2cl-lib:int-sub
        (f2cl-lib:fref inode-%data%
          (ncrnt)
          ((1 *))
          inode-%offset%)
        (f2cl-lib:fref ndimr-%data%
          (il)
          ((1 *))
          ndimr-%offset%)
        1))
(setf (f2cl-lib:fref ndiml-%data% (ir) ((1 *)) ndiml-%offset%)
      (the fixnum
        (truncate
          (f2cl-lib:fref ndimr-%data%
            (ncrnt)
            ((1 *))
            ndimr-%offset%)
          2)))
(setf (f2cl-lib:fref ndimr-%data% (ir) ((1 *)) ndimr-%offset%)
      (f2cl-lib:int-sub
        (f2cl-lib:fref ndimr-%data%
          (ncrnt)
          ((1 *))
          ndimr-%offset%)
        (f2cl-lib:fref ndiml-%data%
          (ir)
          ((1 *))
          ndiml-%offset%)
        1))
(setf (f2cl-lib:fref inode-%data% (ir) ((1 *)) inode-%offset%)
      (f2cl-lib:int-add
        (f2cl-lib:fref inode-%data%
          (ncrnt)
          ((1 *))
          inode-%offset%)
        (f2cl-lib:fref ndiml-%data%
          (ir)
          ((1 *))
          ndiml-%offset%)
        1))))
      (setf llst (f2cl-lib:int-mul llst 2))))
(setf nd (f2cl-lib:int-sub (f2cl-lib:int-mul llst 2) 1))
end_label
(return (values nil lvl nd nil nil nil nil))))))

```

dlaset LAPACK

— dlaset.input —

```
)set break resume
)sys rm -f dlaset.output
)spool dlaset.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlaset.help —

```
=====
dlaset examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASET - an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals

SYNOPSIS

```
SUBROUTINE DLASET( UPLO, M, N, ALPHA, BETA, A, LDA )
```

| | |
|-----------|-----------------------|
| CHARACTER | UPLO |
| INTEGER | LDA, M, N |
| DOUBLE | PRECISION ALPHA, BETA |
| DOUBLE | PRECISION A(LDA, *) |

Purpose

```
=====
```

DLASET initializes an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals.

Arguments

=====

UPLO (input) CHARACTER*1
Specifies the part of the matrix A to be set.
= 'U': Upper triangular part is set; the strictly lower triangular part of A is not changed.
= 'L': Lower triangular part is set; the strictly upper triangular part of A is not changed.
Otherwise: All of the matrix A is set.

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

ALPHA (input) DOUBLE PRECISION
The constant to which the offdiagonal elements are to be set.

BETA (input) DOUBLE PRECISION
The constant to which the diagonal elements are to be set.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On exit, the leading m-by-n submatrix of A is set as follows:

if UPLO = 'U', $A(i,j) = ALPHA$, $1 \leq i \leq j-1$, $1 \leq j \leq n$,
if UPLO = 'L', $A(i,j) = ALPHA$, $j+1 \leq i \leq m$, $1 \leq j \leq n$,
otherwise, $A(i,j) = ALPHA$, $1 \leq i \leq m$, $1 \leq j \leq n$, $i \neq j$,

and, for all UPLO, $A(i,i) = BETA$, $1 \leq i \leq \min(m,n)$.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

— dlaset.f —

```

SUBROUTINE DLASET( UPLO, M, N, ALPHA, BETA, A, LDA )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1992

```

```

*
*   .. Scalar Arguments ..
*   CHARACTER            UPLO
*   INTEGER               LDA, M, N
*   DOUBLE PRECISION      ALPHA, BETA
*
*   ..
*   .. Array Arguments ..
*   DOUBLE PRECISION      A( LDA, * )
*
*   ..
*
*   =====
*
*   .. Local Scalars ..
*   INTEGER               I, J
*
*   ..
*   .. External Functions ..
*   LOGICAL               LSAME
*   EXTERNAL              LSAME
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC             MIN
*
*   ..
*   .. Executable Statements ..
*
*   IF( LSAME( UPLO, 'U' ) ) THEN
*
*       Set the strictly upper triangular or trapezoidal part of the
*       array to ALPHA.
*
*       DO 20 J = 2, N
*           DO 10 I = 1, MIN( J-1, M )
*               A( I, J ) = ALPHA
*   10      CONTINUE
*   20      CONTINUE
*
*   ELSE IF( LSAME( UPLO, 'L' ) ) THEN
*
*       Set the strictly lower triangular or trapezoidal part of the
*       array to ALPHA.
*
*       DO 40 J = 1, MIN( M, N )
*           DO 30 I = J + 1, M
*               A( I, J ) = ALPHA
*   30      CONTINUE
*   40      CONTINUE
*
*   ELSE
*
*       Set the leading m-by-n submatrix to ALPHA.
*

```

```

      DO 60 J = 1, N
        DO 50 I = 1, M
          A( I, J ) = ALPHA
50      CONTINUE
60      CONTINUE
      END IF
*
*      Set the first min(M,N) diagonal elements to BETA.
*
      DO 70 I = 1, MIN( M, N )
        A( I, I ) = BETA
70      CONTINUE
*
      RETURN
*
*      End of DLASET
*
      END

```

— LAPACK dlaset —

```

(defun dlaset (uplo m n alpha beta a lda)
  (declare (type (simple-array double-float (*)) a)
            (type (double-float) beta alpha)
            (type fixnum lda n m)
            (type character uplo))
  (f2cl-lib:with-multi-array-data
    ((uplo character uplo-%data% uplo-%offset%)
     (a double-float a-%data% a-%offset%))
    (prog ((i 0) (j 0))
      (declare (type fixnum j i))
      (cond
        ((char-equal uplo #\U)
         (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
           (> j n) nil)
         (tagbody
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
             (> i
              (min
               (the fixnum
                (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
               (the fixnum m)))
             nil)
           (tagbody
             (setf (f2cl-lib:fref a-%data%
                                   (i j)

```

```

                                ((1 lda) (1 *))
                                a-%offset%)
                                alpha))))))
((char-equal uplo #\L)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j
   (min (the fixnum m)
        (the fixnum n)))
  nil)
 (tagbody
  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
   (> i m) nil)
  (tagbody
   (setf (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)
          alpha))))))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
 (tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
   (> i m) nil)
  (tagbody
   (setf (f2cl-lib:fref a-%data%
                        (i j)
                        ((1 lda) (1 *))
                        a-%offset%)
          alpha))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i
  (min (the fixnum m)
        (the fixnum n)))
  nil)
 (tagbody
  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%) beta)))
(return (values nil nil nil nil nil nil nil))))

```

dlasq1 LAPACK

— dlasq1.input —

)set break resume

```

)sys rm -f dlasq1.output
)spool dlasq1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasq1.help —

```

=====
dlasq1 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASQ1 - the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E

SYNOPSIS

```
SUBROUTINE DLASQ1( N, D, E, WORK, INFO )
```

```
      INTEGER      INFO, N
```

```
      DOUBLE      PRECISION D( * ), E( * ), WORK( * )
```

Purpose

```
=====
```

DLASQ1 computes the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E. The singular values are computed to high relative accuracy, in the absence of denormalization, underflow and overflow. The algorithm was first presented in

"Accurate singular values and differential qd algorithms" by K. V. Fernando and B. N. Parlett, Numer. Math., Vol-67, No. 2, pp. 191-230, 1994,

and the present implementation is described in "An implementation of the dqds Algorithm (Positive Case)", LAPACK Working Note.

Arguments

=====

N (input) INTEGER
 The number of rows and columns in the matrix. N >= 0.

D (input/output) DOUBLE PRECISION array, dimension (N)
 On entry, D contains the diagonal elements of the
 bidiagonal matrix whose SVD is desired. On normal exit,
 D contains the singular values in decreasing order.

E (input/output) DOUBLE PRECISION array, dimension (N)
 On entry, elements E(1:N-1) contain the off-diagonal elements
 of the bidiagonal matrix whose SVD is desired.
 On exit, E is overwritten.

WORK (workspace) DOUBLE PRECISION array, dimension (4*N)

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value
 > 0: the algorithm failed
 = 1, a split was marked by a positive value in E
 = 2, current block of Z not diagonalized after 30*N
 iterations (in inner while loop)
 = 3, termination criterion of outer while loop not met
 (program created more than N unreduced blocks)

— dlasq1.f —

```

SUBROUTINE DLASQ1( N, D, E, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
*  INTEGER          INFO, N
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION D( * ), E( * ), WORK( * )
*  ..
*
*  =====
*
*  .. Parameters ..

```

```

      DOUBLE PRECISION  ZERO
      PARAMETER          ( ZERO = 0.0D0 )
*
*      ..
*      .. Local Scalars ..
      INTEGER            I, IINFO
      DOUBLE PRECISION   EPS, SCALE, SAFMIN, SIGMN, SIGMX
*
*      ..
*      .. External Subroutines ..
      EXTERNAL           DLAS2, DLASQ2, DLASRT, XERBLA
*
*      ..
*      .. External Functions ..
      DOUBLE PRECISION   DLAMCH
      EXTERNAL           DLAMCH
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC           ABS, MAX, SQRT
*
*      ..
*      .. Executable Statements ..
*
      INFO = 0
      IF( N.LT.0 ) THEN
        INFO = -2
        CALL XERBLA( 'DLASQ1', -INFO )
        RETURN
      ELSE IF( N.EQ.0 ) THEN
        RETURN
      ELSE IF( N.EQ.1 ) THEN
        D( 1 ) = ABS( D( 1 ) )
        RETURN
      ELSE IF( N.EQ.2 ) THEN
        CALL DLAS2( D( 1 ), E( 1 ), D( 2 ), SIGMN, SIGMX )
        D( 1 ) = SIGMX
        D( 2 ) = SIGMN
        RETURN
      END IF

*
*      Estimate the largest singular value.
*
      SIGMX = ZERO
      DO 10 I = 1, N - 1
        D( I ) = ABS( D( I ) )
        SIGMX = MAX( SIGMX, ABS( E( I ) ) )
10 CONTINUE
      D( N ) = ABS( D( N ) )

*
*      Early return if SIGMX is zero (matrix is already diagonal).
*
      IF( SIGMX.EQ.ZERO ) THEN
        CALL DLASRT( 'D', N, D, IINFO )
        RETURN

```

```

      END IF
*
      DO 20 I = 1, N
          SIGMX = MAX( SIGMX, D( I ) )
20  CONTINUE
*
*      Copy D and E into WORK (in the Z format) and scale (squaring the
*      input data makes scaling by a power of the radix pointless).
*
      EPS = DLAMCH( 'Precision' )
      SAFMIN = DLAMCH( 'Safe minimum' )
      SCALE = SQRT( EPS / SAFMIN )
      CALL DCOPY( N, D, 1, WORK( 1 ), 2 )
      CALL DCOPY( N-1, E, 1, WORK( 2 ), 2 )
      CALL DLASCL( 'G', 0, 0, SIGMX, SCALE, 2*N-1, 1, WORK, 2*N-1,
$          IINFO )
*
*      Compute the q's and e's.
*
      DO 30 I = 1, 2*N - 1
          WORK( I ) = WORK( I )**2
30  CONTINUE
      WORK( 2*N ) = ZERO
*
      CALL DLASQ2( N, WORK, INFO )
*
      IF( INFO.EQ.0 ) THEN
          DO 40 I = 1, N
              D( I ) = SQRT( WORK( I ) )
40  CONTINUE
          CALL DLASCL( 'G', 0, 0, SCALE, SIGMX, N, 1, D, N, IINFO )
      END IF
*
      RETURN
*
*      End of DLASQ1
*
      END

```

— LAPACK dlasq1 —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq1 (n d e work info)
    (declare (type (simple-array double-float (*)) work e d)
              (type fixnum info n))

```

```

(f2cl-lib:with-multi-array-data
  ((d double-float d-%data% d-%offset%)
   (e double-float e-%data% e-%offset%)
   (work double-float work-%data% work-%offset%))
  (prog ((eps 0.0) (scale 0.0) (safmin 0.0) (sigmn 0.0) (sigmx 0.0) (i 0)
        (iinfo 0))
    (declare (type (double-float) eps scale safmin sigmn sigmx)
              (type fixnum i iinfo))
    (setf info 0)
    (cond
      ((< n 0)
       (setf info -2)
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DLASQ1" (f2cl-lib:int-sub info))
       (go end_label))
      (= n 0)
      (go end_label))
      ((= n 1)
       (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
              (abs (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)))
       (go end_label))
      ((= n 2)
       (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
         (dlas2 (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%)
                 (f2cl-lib:fref e-%data% (1) ((1 *)) e-%offset%)
                 (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%) sigmn sigmx)
         (declare (ignore var-0 var-1 var-2))
         (setf sigmn var-3)
         (setf sigmx var-4)
         (setf (f2cl-lib:fref d-%data% (1) ((1 *)) d-%offset%) sigmx)
         (setf (f2cl-lib:fref d-%data% (2) ((1 *)) d-%offset%) sigmn)
         (go end_label)))
       (setf sigmx zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
         (tagbody
          (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
                 (abs (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)))
          (setf sigmx
                 (max sigmx
                      (abs
                       (f2cl-lib:fref e-%data% (i) ((1 *)) e-%offset%))))))
       (setf (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)
              (abs (f2cl-lib:fref d-%data% (n) ((1 *)) d-%offset%)))
      (cond
        ((= sigmx zero)
         (multiple-value-bind (var-0 var-1 var-2 var-3)
           (dlasrt "D" n d iinfo)
           (declare (ignore var-0 var-1 var-2))

```

```

        (setf iinfo var-3))
      (go end_label)))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf sigmx
        (max sigmx
          (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))))
    (setf eps (dlamch "Precision"))
    (setf safmin (dlamch "Safe minimum"))
    (setf scale (f2cl-lib:fsqrt (/ eps safmin)))
    (dcopy n d 1 (f2cl-lib:array-slice work double-float (1) ((1 *))) 2)
    (dcopy (f2cl-lib:int-sub n 1) e 1
      (f2cl-lib:array-slice work double-float (2) ((1 *))) 2)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (dlascl "G" 0 0 sigmx scale
        (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1) 1 work
        (f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1) iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8)))
    (setf iinfo var-9))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
          (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
        (expt (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
          2))))
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-mul 2 n)
        ((1 *))
        work-%offset%)
      zero)
    (multiple-value-bind (var-0 var-1 var-2)
      (dlasq2 n work info)
      (declare (ignore var-0 var-1))
      (setf info var-2))
    (cond
      ((= info 0)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
            (f2cl-lib:fsqrt
              (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%))))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)

```

```

        (dlascl "G" 0 0 scale sigmx n 1 d n iinfo)
        (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                        var-8))
        (setf iinfo var-9))))
end_label
(return (values nil nil nil nil info))))))

```

dlasq2 LAPACK

— dlasq2.input —

```

)set break resume
)sys rm -f dlasq2.output
)spool dlasq2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasq2.help —

```

=====
dlasq2 examples
=====

=====
Man Page Details
=====

```

NAME

DLASQ2 - all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z to high relative accuracy are computed to high relative accuracy, in the absence of denormalization, underflow and overflow

SYNOPSIS

```
SUBROUTINE DLASQ2( N, Z, INFO )
```

```

        INTEGER          INFO, N

```

DOUBLE PRECISION Z(*)

Purpose

=====

DLASQ2 computes all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z to high relative accuracy are computed to high relative accuracy, in the absence of denormalization, underflow and overflow.

To see the relation of Z to the tridiagonal matrix, let L be a unit lower bidiagonal matrix with subdiagonals Z(2,4,6,...) and let U be an upper bidiagonal matrix with 1's above and diagonal Z(1,3,5,...). The tridiagonal is L*U or, if you prefer, the symmetric tridiagonal to which it is similar.

Note : DLASQ2 defines a logical variable, IEEE, which is true on machines which follow ieee-754 floating-point standard in their handling of infinities and NaNs, and false otherwise. This variable is passed to DLASQ3.

Arguments

=====

N (input) INTEGER
 The number of rows and columns in the matrix. N >= 0.

Z (workspace) DOUBLE PRECISION array, dimension (4*N)
 On entry Z holds the qd array. On exit, entries 1 to N hold the eigenvalues in decreasing order, Z(2*N+1) holds the trace, and Z(2*N+2) holds the sum of the eigenvalues. If N > 2, then Z(2*N+3) holds the iteration count, Z(2*N+4) holds NDIVS/NIN², and Z(2*N+5) holds the percentage of shifts that failed.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if the i-th argument is a scalar and had an illegal value, then INFO = -i, if the i-th argument is an array and the j-entry had an illegal value, then INFO = -(i*100+j)
 > 0: the algorithm failed
 = 1, a split was marked by a positive value in E
 = 2, current block of Z not diagonalized after 30*N iterations (in inner while loop)
 = 3, termination criterion of outer while loop not met (program created more than N unreduced blocks)

Further Details

=====

Local Variables: IO:NO defines a current unreduced segment of Z.
The shifts are accumulated in SIGMA. Iteration count is in ITER.
Ping-pong is controlled by PP (alternates between 0 and 1).

— dlasq2.f —

```

SUBROUTINE DLASQ2( N, Z, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1999
*
*    .. Scalar Arguments ..
      INTEGER          INFO, N
*
*    ..
*
*    .. Array Arguments ..
      DOUBLE PRECISION Z( * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
      DOUBLE PRECISION CBIAS
      PARAMETER          ( CBIAS = 1.50D0 )
      DOUBLE PRECISION ZERO, HALF, ONE, TWO, FOUR, HUNDRD
      PARAMETER          ( ZERO = 0.0D0, HALF = 0.5D0, ONE = 1.0D0,
$                        TWO = 2.0D0, FOUR = 4.0D0, HUNDRD = 100.0D0 )
*
*    ..
*
*    .. Local Scalars ..
      LOGICAL            IEEE
      INTEGER            IO, I4, IINFO, IPN4, ITER, IWHILA, IWHILB, K,
$                      NO, NBIG, NDIV, NFAIL, PP, SPLT
      DOUBLE PRECISION  D, DESIG, DMIN, E, EMAX, EMIN, EPS, OLDEMN,
$                      QMAX, QMIN, S, SAFMIN, SIGMA, T, TEMP, TOL,
$                      TOL2, TRACE, ZMAX
*
*    ..
*
*    .. External Subroutines ..
      EXTERNAL           DLASQ3, DLASRT, XERBLA
*
*    ..
*
*    .. External Functions ..
      INTEGER            ILAENV
      DOUBLE PRECISION  DLAMCH
      EXTERNAL           DLAMCH, ILAENV
*
*    ..

```



```

*      .. Intrinsic Functions ..
      INTRINSIC          DBLE, MAX, MIN, SQRT
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments.
*      (in case DLASQ2 is not called by DLASQ1)
*
      INFO = 0
      EPS = DLAMCH( 'Precision' )
      SAFMIN = DLAMCH( 'Safe minimum' )
      TOL = EPS*HUNDRD
      TOL2 = TOL**2
*
      IF( N.LT.0 ) THEN
        INFO = -1
        CALL XERBLA( 'DLASQ2', 1 )
        RETURN
      ELSE IF( N.EQ.0 ) THEN
        RETURN
      ELSE IF( N.EQ.1 ) THEN
*
*      1-by-1 case.
*
        IF( Z( 1 ).LT.ZERO ) THEN
          INFO = -201
          CALL XERBLA( 'DLASQ2', 2 )
        END IF
        RETURN
      ELSE IF( N.EQ.2 ) THEN
*
*      2-by-2 case.
*
        IF( Z( 2 ).LT.ZERO .OR. Z( 3 ).LT.ZERO ) THEN
          INFO = -2
          CALL XERBLA( 'DLASQ2', 2 )
          RETURN
        ELSE IF( Z( 3 ).GT.Z( 1 ) ) THEN
          D = Z( 3 )
          Z( 3 ) = Z( 1 )
          Z( 1 ) = D
        END IF
        Z( 5 ) = Z( 1 ) + Z( 2 ) + Z( 3 )
        IF( Z( 2 ).GT.Z( 3 )*TOL2 ) THEN
          T = HALF*( ( Z( 1 )-Z( 3 ) )+Z( 2 ) )
          S = Z( 3 )*( Z( 2 ) / T )
          IF( S.LE.T ) THEN
            S = Z( 3 )*( Z( 2 ) / ( T*( ONE+SQRT( ONE+S / T ) ) ) )
          ELSE
            S = Z( 3 )*( Z( 2 ) / ( T+SQRT( T )*SQRT( T+S ) ) )
          END IF
        END IF
      END IF

```

```

        END IF
        T = Z( 1 ) + ( S+Z( 2 ) )
        Z( 3 ) = Z( 3 )*( Z( 1 ) / T )
        Z( 1 ) = T
    END IF
    Z( 2 ) = Z( 3 )
    Z( 6 ) = Z( 2 ) + Z( 1 )
    RETURN
END IF

*
*   Check for negative data and compute sums of q's and e's.
*
    Z( 2*N ) = ZERO
    EMIN = Z( 2 )
    QMAX = ZERO
    ZMAX = ZERO
    D = ZERO
    E = ZERO

*
    DO 10 K = 1, 2*( N-1 ), 2
        IF( Z( K ).LT.ZERO ) THEN
            INFO = -( 200+K )
            CALL XERBLA( 'DLASQ2', 2 )
            RETURN
        ELSE IF( Z( K+1 ).LT.ZERO ) THEN
            INFO = -( 200+K+1 )
            CALL XERBLA( 'DLASQ2', 2 )
            RETURN
        END IF
        D = D + Z( K )
        E = E + Z( K+1 )
        QMAX = MAX( QMAX, Z( K ) )
        EMIN = MIN( EMIN, Z( K+1 ) )
        ZMAX = MAX( QMAX, ZMAX, Z( K+1 ) )
    10 CONTINUE
    IF( Z( 2*N-1 ).LT.ZERO ) THEN
        INFO = -( 200+2*N-1 )
        CALL XERBLA( 'DLASQ2', 2 )
        RETURN
    END IF
    D = D + Z( 2*N-1 )
    QMAX = MAX( QMAX, Z( 2*N-1 ) )
    ZMAX = MAX( QMAX, ZMAX )

*
*   Check for diagonality.
*
    IF( E.EQ.ZERO ) THEN
        DO 20 K = 2, N
            Z( K ) = Z( 2*K-1 )
        20 CONTINUE

```

```

        CALL DLASRT( 'D', N, Z, IINFO )
        Z( 2*N-1 ) = D
        RETURN
    END IF
*
    TRACE = D + E
*
*   Check for zero data.
*
    IF( TRACE.EQ.ZERO ) THEN
        Z( 2*N-1 ) = ZERO
        RETURN
    END IF
*
*   Check whether the machine is IEEE conformable.
*
    IEEE = ILAENV( 10, 'DLASQ2', 'N', 1, 2, 3, 4 ).EQ.1 .AND.
$      ILAENV( 11, 'DLASQ2', 'N', 1, 2, 3, 4 ).EQ.1
*
*   Rearrange data for locality: Z=(q1,qq1,e1,ee1,q2,qq2,e2,ee2,...).
*
    DO 30 K = 2*N, 2, -2
        Z( 2*K ) = ZERO
        Z( 2*K-1 ) = Z( K )
        Z( 2*K-2 ) = ZERO
        Z( 2*K-3 ) = Z( K-1 )
30    CONTINUE
*
    IO = 1
    NO = N
*
*   Reverse the qd-array, if warranted.
*
    IF( CBIAS*Z( 4*IO-3 ).LT.Z( 4*NO-3 ) ) THEN
        IPN4 = 4*( IO+NO )
        DO 40 I4 = 4*IO, 2*( IO+NO-1 ), 4
            TEMP = Z( I4-3 )
            Z( I4-3 ) = Z( IPN4-I4-3 )
            Z( IPN4-I4-3 ) = TEMP
            TEMP = Z( I4-1 )
            Z( I4-1 ) = Z( IPN4-I4-5 )
            Z( IPN4-I4-5 ) = TEMP
40    CONTINUE
        END IF
*
*   Initial split checking via dqd and Li's test.
*
    PP = 0
*
    DO 80 K = 1, 2

```

```

*
      D = Z( 4*NO+PP-3 )
      DO 50 I4 = 4*( NO-1 ) + PP, 4*IO + PP, -4
        IF( Z( I4-1 ).LE.TOL2*D ) THEN
          Z( I4-1 ) = -ZERO
          D = Z( I4-3 )
        ELSE
          D = Z( I4-3 )*( D / ( D+Z( I4-1 ) ) )
        END IF
50    CONTINUE
*
*      dqd maps Z to ZZ plus Li's test.
*
      EMIN = Z( 4*IO+PP+1 )
      D = Z( 4*IO+PP-3 )
      DO 60 I4 = 4*IO + PP, 4*( NO-1 ) + PP, 4
        Z( I4-2*PP-2 ) = D + Z( I4-1 )
        IF( Z( I4-1 ).LE.TOL2*D ) THEN
          Z( I4-1 ) = -ZERO
          Z( I4-2*PP-2 ) = D
          Z( I4-2*PP ) = ZERO
          D = Z( I4+1 )
        ELSE IF( SAFMIN*Z( I4+1 ).LT.Z( I4-2*PP-2 ) .AND.
$          SAFMIN*Z( I4-2*PP-2 ).LT.Z( I4+1 ) ) THEN
          TEMP = Z( I4+1 ) / Z( I4-2*PP-2 )
          Z( I4-2*PP ) = Z( I4-1 )*TEMP
          D = D*TEMP
        ELSE
          Z( I4-2*PP ) = Z( I4+1 )*( Z( I4-1 ) / Z( I4-2*PP-2 ) )
          D = Z( I4+1 )*( D / Z( I4-2*PP-2 ) )
        END IF
        EMIN = MIN( EMIN, Z( I4-2*PP ) )
60    CONTINUE
      Z( 4*NO-PP-2 ) = D
*
*      Now find qmax.
*
      QMAX = Z( 4*IO-PP-2 )
      DO 70 I4 = 4*IO - PP + 2, 4*NO - PP - 2, 4
        QMAX = MAX( QMAX, Z( I4 ) )
70    CONTINUE
*
*      Prepare for the next iteration on K.
*
      PP = 1 - PP
80 CONTINUE
*
      ITER = 2
      NFAIL = 0
      NDIV = 2*( NO-IO )

```

```

*
      DO 140 IWHILA = 1, N + 1
          IF( NO.LT.1 )
$           GO TO 150
*
*       While array unfinished do
*
*       E(NO) holds the value of SIGMA when submatrix in IO:NO
*       splits from the rest of the array, but is negated.
*
      DESIG = ZERO
      IF( NO.EQ.N ) THEN
          SIGMA = ZERO
      ELSE
          SIGMA = -Z( 4*NO-1 )
      END IF
      IF( SIGMA.LT.ZERO ) THEN
          INFO = 1
          RETURN
      END IF
*
*       Find last unreduced submatrix's top index IO, find QMAX and
*       EMIN. Find Gershgorin-type bound if Q's much greater than E's.
*
      EMAX = ZERO
      IF( NO.GT.IO ) THEN
          EMIN = ABS( Z( 4*NO-5 ) )
      ELSE
          EMIN = ZERO
      END IF
      QMIN = Z( 4*NO-3 )
      QMAX = QMIN
      DO 90 I4 = 4*NO, 8, -4
          IF( Z( I4-5 ).LE.ZERO )
$           GO TO 100
          IF( QMIN.GE.FOUR*EMAX ) THEN
              QMIN = MIN( QMIN, Z( I4-3 ) )
              EMAX = MAX( EMAX, Z( I4-5 ) )
          END IF
          QMAX = MAX( QMAX, Z( I4-7 )+Z( I4-5 ) )
          EMIN = MIN( EMIN, Z( I4-5 ) )
90      CONTINUE
      I4 = 4
*
100     CONTINUE
      IO = I4 / 4
*
*       Store EMIN for passing to DLASQ3.
*
      Z( 4*NO-1 ) = EMIN

```

```

*
*      Put -(initial shift) into DMIN.
*
*      DMIN = -MAX( ZERO, QMIN-TWO*SQRT( QMIN )*SQRT( EMAX ) )
*
*      Now IO:NO is unreduced. PP = 0 for ping, PP = 1 for pong.
*
*      PP = 0
*
*      NBIG = 30*( NO-IO+1 )
*      DO 120 IWHILB = 1, NBIG
*          IF( IO.GT.NO )
*              $          GO TO 130
*
*      While submatrix unfinished take a good dqds step.
*
*      CALL DLASQ3( IO, NO, Z, PP, DMIN, SIGMA, DESIG, QMAX, NFAIL,
*          $          ITER, NDIV, IEEE )
*
*      PP = 1 - PP
*
*      When EMIN is very small check for splits.
*
*      IF( PP.EQ.0 .AND. NO-IO.GE.3 ) THEN
*          IF( Z( 4*NO ).LE.TOL2*QMAX .OR.
*              $          Z( 4*NO-1 ).LE.TOL2*SIGMA ) THEN
*              SPLT = IO - 1
*              QMAX = Z( 4*IO-3 )
*              EMIN = Z( 4*IO-1 )
*              OLDEMN = Z( 4*IO )
*              DO 110 I4 = 4*IO, 4*( NO-3 ), 4
*                  IF( Z( I4 ).LE.TOL2*Z( I4-3 ) .OR.
*                      $          Z( I4-1 ).LE.TOL2*SIGMA ) THEN
*                      Z( I4-1 ) = -SIGMA
*                      SPLT = I4 / 4
*                      QMAX = ZERO
*                      EMIN = Z( I4+3 )
*                      OLDEMN = Z( I4+4 )
*                  ELSE
*                      QMAX = MAX( QMAX, Z( I4+1 ) )
*                      EMIN = MIN( EMIN, Z( I4-1 ) )
*                      OLDEMN = MIN( OLDEMN, Z( I4 ) )
*                  END IF
*              110      CONTINUE
*              Z( 4*NO-1 ) = EMIN
*              Z( 4*NO ) = OLDEMN
*              IO = SPLT + 1
*          END IF
*      END IF
*
*

```

```

120    CONTINUE
*
        INFO = 2
        RETURN
*
*    end IWHILB
*
130    CONTINUE
*
140 CONTINUE
*
        INFO = 3
        RETURN
*
*    end IWHILA
*
150 CONTINUE
*
*    Move q's to the front.
*
        DO 160 K = 2, N
            Z( K ) = Z( 4*K-3 )
160 CONTINUE
*
*    Sort and compute sum of eigenvalues.
*
        CALL DLASRT( 'D', N, Z, IINFO )
*
        E = ZERO
        DO 170 K = N, 1, -1
            E = E + Z( K )
170 CONTINUE
*
*    Store trace, sum(eigenvalues) and information on performance.
*
        Z( 2*N+1 ) = TRACE
        Z( 2*N+2 ) = E
        Z( 2*N+3 ) = DBLE( ITER )
        Z( 2*N+4 ) = DBLE( NDIV ) / DBLE( N**2 )
        Z( 2*N+5 ) = HUNDRD*NFAIL / DBLE( ITER )
        RETURN
*
*    End of DLASQ2
*
END

```

— LAPACK dlasq2 —

```

(let* ((cbias 1.5)
      (zero 0.0)
      (half 0.5)
      (one 1.0)
      (two 2.0)
      (four 4.0)
      (hundrd 100.0))
  (declare (type (double-float 1.5 1.5) cbias)
           (type (double-float 0.0 0.0) zero)
           (type (double-float 0.5 0.5) half)
           (type (double-float 1.0 1.0) one)
           (type (double-float 2.0 2.0) two)
           (type (double-float 4.0 4.0) four)
           (type (double-float 100.0 100.0) hundrd))
  (defun dlasq2 (n z info)
    (declare (type (simple-array double-float (*)) z)
             (type fixnum info n))
    (f2cl-lib:with-multi-array-data
      ((z double-float z-%data% z-%offset%))
      (prog ((d 0.0) (desig 0.0) (dmin 0.0) (e 0.0) (emax 0.0) (emin 0.0)
            (eps 0.0) (oldemn 0.0) (qmax 0.0) (qmin 0.0) (s 0.0) (safmin 0.0)
            (sigma 0.0) (temp 0.0) (tol 0.0) (tol2 0.0) (zmax 0.0) (i0 0)
            (i4 0) (iinfo 0) (ipn4 0) (iter 0) (iwhila 0) (iwhilb 0) (k 0)
            (n0 0) (nbig 0) (ndiv 0) (nfail 0) (pp 0) (spltt 0) (ieee nil)
            (trace$ 0.0) (t$ 0.0))
      (declare (type (double-float) t$ trace$ d desig dmin e emax emin eps
                    oldemn qmax qmin s safmin sigma temp tol
                    tol2 zmax)
               (type fixnum i0 i4 iinfo ipn4 iter iwhila iwhilb
                    k n0 nbig ndiv nfail pp spltt)
               (type (member t nil) iieee))
      (setf info 0)
      (setf eps (dlamch "Precision"))
      (setf safmin (dlamch "Safe minimum"))
      (setf tol (* eps hundrd))
      (setf tol2 (expt tol 2))
      (cond
        ((< n 0)
         (setf info -1)
         (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DLASQ2" 1)
         (go end_label))
        (= n 0)
        (go end_label))
      ((= n 1)
       (cond
        ((< (f2cl-lib:fref z (1) ((1 *))) zero)

```



```

(setf info -201)
(error
 " ** On entry to ~a parameter number ~a had an illegal value~%"
 "DLASQ2" 2)))
(go end_label))
(= n 2)
(cond
 ((or (< (f2cl-lib:fref z (2) ((1 *))) zero)
 (< (f2cl-lib:fref z (3) ((1 *))) zero))
 (setf info -2)
 (error
 " ** On entry to ~a parameter number ~a had an illegal value~%"
 "DLASQ2" 2)
 (go end_label))
 ((> (f2cl-lib:fref z (3) ((1 *))) (f2cl-lib:fref z (1) ((1 *))))
 (setf d (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%))
 (setf (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
 (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%))
 (setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) d)))
(setf (f2cl-lib:fref z-%data% (5) ((1 *)) z-%offset%)
 (+ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
 (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
 (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)))
(cond
 ((> (f2cl-lib:fref z (2) ((1 *)))
 (* (f2cl-lib:fref z (3) ((1 *))) tol2))
 (setf t$
 (* half
 (+
 (- (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
 (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
 (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))))))
 (setf s
 (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
 (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
 t$)))
(cond
 ((<= s t$)
 (setf s
 (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
 (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
 (* t$
 (+ one (f2cl-lib:fsqrt (+ one (/ s t$))))))))))
(t
 (setf s
 (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
 (/ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
 (+ t$
 (* (f2cl-lib:fsqrt t$)
 (f2cl-lib:fsqrt (+ t$ s))))))))))

```

```

(setf t$
  (+ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
     (+ s (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))))
(setf (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%)
         (/ (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)
            t$)))
(setf (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%) t$))
(setf (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data% (3) ((1 *)) z-%offset%))
(setf (f2cl-lib:fref z-%data% (6) ((1 *)) z-%offset%)
      (+ (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%)
         (f2cl-lib:fref z-%data% (1) ((1 *)) z-%offset%)))
(go end_label)))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-mul 2 n))
                    ((1 *))
                    z-%offset%)
      zero)
(setf emin (f2cl-lib:fref z-%data% (2) ((1 *)) z-%offset%))
(setf qmax zero)
(setf zmax zero)
(setf d zero)
(setf e zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 2))
              (> k
               (f2cl-lib:int-mul 2
                                   (f2cl-lib:int-add n
                                                       (f2cl-lib:int-sub
                                                         1))))
              nil)
(tagbody
 (cond
  ((< (f2cl-lib:fref z (k) ((1 *))) zero)
   (setf info (f2cl-lib:int-sub (f2cl-lib:int-add 200 k)))
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASQ2" 2)
   (go end_label))
  ((< (f2cl-lib:fref z ((f2cl-lib:int-add k 1)) ((1 *))) zero)
   (setf info (f2cl-lib:int-sub (f2cl-lib:int-add 200 k 1)))
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASQ2" 2)
   (go end_label)))
(setf d (+ d (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)))
(setf e
  (+ e
    (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-add k 1))

```

```

                                ((1 *))
                                z-%offset%)))

(setf qmax
  (max qmax (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add k 1))
      ((1 *))
      z-%offset%)))

(setf zmax
  (max qmax
    zmax
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add k 1))
      ((1 *))
      z-%offset%))))))

(cond
  ((<
    (f2cl-lib:fref z
      ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n)
        (f2cl-lib:int-sub 1)))
      ((1 *)))
    zero)
  (setf info
    (f2cl-lib:int-sub
      (f2cl-lib:int-sub
        (f2cl-lib:int-add 200 (f2cl-lib:int-mul 2 n))
        1)))
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DLASQ2" 2)
  (go end_label)))

(setf d
  (+ d
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
      ((1 *))
      z-%offset%)))

(setf qmax
  (max qmax
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n)
        1))
      ((1 *))
      z-%offset%)))

(setf zmax (max qmax zmax))
(cond
  ((= e zero)
    (f2cl-lib:fdo (k 2 (f2cl-lib:int-add k 1))

```

```
((> k n) nil)
(tagbody
  (setf (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k)
        1))
      ((1 *))
      z-%offset%))))
(multiple-value-bind (var-1 var-2 var-3)
  (dlasrt "D" n z iinfo)
  (declare (ignore var-0 var-1 var-2))
  (setf iinfo var-3))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
  ((1 *))
  z-%offset%)
  d)
(go end_label)))
(setf trace$ (+ d e))
(cond
  (= trace$ zero)
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 n) 1))
    ((1 *))
    z-%offset%)
    zero)
  (go end_label)))
(setf ieee
  (and (= (ilaenv 10 "DLASQ2" "N" 1 2 3 4) 1)
    (= (ilaenv 11 "DLASQ2" "N" 1 2 3 4) 1)))
(f2cl-lib:fdo (k (f2cl-lib:int-mul 2 n)
  (f2cl-lib:int-add k (f2cl-lib:int-sub 2)))
  ((> k 2) nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-mul 2 k))
      ((1 *))
      z-%offset%)
      zero)
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 1))
      ((1 *))
      z-%offset%)
      (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%))
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 2))
      ((1 *))
      z-%offset%)
      zero)
    (setf (f2cl-lib:fref z-%data%
```

```

((f2cl-lib:int-sub (f2cl-lib:int-mul 2 k) 3))
((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub k 1))
  ((1 *))
  z-%offset%))))))
(setf i0 1)
(setf n0 n)
(cond
  ((<
    (* cbias
      (f2cl-lib:fref z
        ((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
          (f2cl-lib:int-sub 3)))
        ((1 *))))))
    (f2cl-lib:fref z
      ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
        (f2cl-lib:int-sub 3)))
      ((1 *))))
    (setf ipn4 (f2cl-lib:int-mul 4 (f2cl-lib:int-add i0 n0)))
    (f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add i4 4))
      (> i4
        (f2cl-lib:int-mul 2
          (f2cl-lib:int-add i0
            n0
            (f2cl-lib:int-sub
              1))))))
    nil)
  (tagbody
    (setf temp
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub i4 3))
        ((1 *))
        z-%offset%))
      (setf (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub i4 3))
        ((1 *))
        z-%offset%))
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub ipn4 i4 3))
          ((1 *))
          z-%offset%))
        (setf (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub ipn4 i4 3))
          ((1 *))
          z-%offset%))
          temp)
      (setf temp
        (f2cl-lib:fref z-%data%

```

```

((f2cl-lib:int-sub i4 1))
((1 *))
z-%offset%)
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub i4 1))
((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 i4 5))
((1 *))
z-%offset%))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 i4 5))
((1 *))
z-%offset%))
temp))))
(setf pp 0)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
(> k 2) nil)
(tagbody
(setf d
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
pp)
3))
((1 *))
z-%offset%))
(f2cl-lib:fdo (i4
(f2cl-lib:int-add
(f2cl-lib:int-mul 4
(f2cl-lib:int-add n0
(f2cl-lib:int-sub
1))))
pp)
(f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
(> i4 (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp))
nil)
(tagbody
(cond
(<=
(f2cl-lib:fref z
((f2cl-lib:int-add i4 (f2cl-lib:int-sub 1)))
((1 *)))
(* tol2 d))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub i4 1))
((1 *))
z-%offset%)
(- zero))

```

```

      (setf d
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub i4 3))
          ((1 *))
          z-%offset%)))
    (t
      (setf d
        (*
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub i4 3))
            ((1 *))
            z-%offset%)
          (/ d
            (+ d
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub i4 1))
                ((1 *))
                z-%offset%))))))))))
    (setf emin
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
          pp
          1))
        ((1 *))
        z-%offset%))
    (setf d
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
            pp)
          3))
        ((1 *))
        z-%offset%))
    (f2cl-lib:fdo (i4 (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
      (f2cl-lib:int-add i4 4))
      (> i4
        (f2cl-lib:int-add
          (f2cl-lib:int-mul 4
            (f2cl-lib:int-add n0
              (f2cl-lib:int-sub
                1)))
          pp))
      nil)
    (tagbody
      (setf (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-add i4
            (f2cl-lib:int-mul -1
              2
              pp))

```

```

                2))
                ((1 *))
                z-%offset%)
(+ d
  (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub i4 1))
    ((1 *))
    z-%offset%)))
(cond
  ((<=
    (f2cl-lib:fref z
      ((f2cl-lib:int-add i4 (f2cl-lib:int-sub 1)))
      ((1 *)))
    (* tol2 d))
   (setf (f2cl-lib:fref z-%data%
     ((f2cl-lib:int-sub i4 1))
     ((1 *))
     z-%offset%)
     (- zero))
   (setf (f2cl-lib:fref z-%data%
     ((f2cl-lib:int-sub
       (f2cl-lib:int-add i4
         (f2cl-lib:int-mul
           -1
           2
           pp))
       2))
     ((1 *))
     z-%offset%)
     d)
   (setf (f2cl-lib:fref z-%data%
     ((f2cl-lib:int-add i4
       (f2cl-lib:int-mul -1
        2
        pp)))
     ((1 *))
     z-%offset%)
     zero)
   (setf d
     (f2cl-lib:fref z-%data%
       ((f2cl-lib:int-add i4 1))
       ((1 *))
       z-%offset%)))
  ((and
    (<
      (* safmin
        (f2cl-lib:fref z ((f2cl-lib:int-add i4 1)) ((1 *))))
      (f2cl-lib:fref z
        ((f2cl-lib:int-add i4
          (f2cl-lib:int-mul -1
```



```

                                2
                                pp)
                                (f2cl-lib:int-sub 2)))
                                ((1 *)))
(<
  (* safmin
    (f2cl-lib:fref z
      ((f2cl-lib:int-add i4
        (f2cl-lib:int-mul -1
          2
          pp)
        (f2cl-lib:int-sub
          2)))
      ((1 *)))
    (f2cl-lib:fref z ((f2cl-lib:int-add i4 1)) ((1 *))))
(setf temp
  (/
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add i4 1))
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub
        (f2cl-lib:int-add i4
          (f2cl-lib:int-mul
            -1
            2
            pp))
          2))
      ((1 *))
      z-%offset%)))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-add i4
    (f2cl-lib:int-mul -1
      2
      pp)))
  ((1 *))
  z-%offset%)
  (*
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub i4 1))
      ((1 *))
      z-%offset%)
    temp))
(setf d (* d temp)))
(t
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-add i4
      (f2cl-lib:int-mul -1
        2

```

```

                                                    pp)))
                    ((1 *))
                    z-%offset%)
(*
(f2cl-lib:fref z-%data%
  ((f2cl-lib:int-add i4 1))
  ((1 *))
  z-%offset%)
(/
(f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub i4 1))
  ((1 *))
  z-%offset%)
(f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add i4
      (f2cl-lib:int-mul
        -1
        2
        pp))
    2))
  ((1 *))
  z-%offset%))))))
(setf d
  (*
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add i4 1))
      ((1 *))
      z-%offset%)
    (/ d
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-add i4
            (f2cl-lib:int-mul
              -1
              2
              pp))
          2))
        ((1 *))
        z-%offset%))))))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add i4
        (f2cl-lib:int-mul
          -1
          2
          pp))
      ((1 *))
      z-%offset%))))))

```

```

(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
                                         pp
                                         2))
                    ((1 *))
                    z-%offset%))
d)
(setf qmax
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 i0)
                                         pp
                                         2))
                    ((1 *))
                    z-%offset%))
(f2cl-lib:fdo (i4
              (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
                                (f2cl-lib:int-sub pp)
                                2)
              (f2cl-lib:int-add i4 4))
              ((> i4
                (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                  (f2cl-lib:int-sub pp)
                                  (f2cl-lib:int-sub 2)))
              nil)
      (tagbody
        (setf qmax
              (max qmax
                    (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%))))
        (setf pp (f2cl-lib:int-sub 1 pp)))
(setf iter 2)
(setf nfail 0)
(setf ndiv (f2cl-lib:int-mul 2 (f2cl-lib:int-sub n0 i0)))
(f2cl-lib:fdo (iwhila 1 (f2cl-lib:int-add iwhila 1))
              ((> iwhila (f2cl-lib:int-add n 1)) nil)
              (tagbody
                (if (< n0 1) (go label150))
                (setf desig zero)
                (cond
                  ((= n0 n)
                   (setf sigma zero))
                  (t
                   (setf sigma
                         (-
                          (f2cl-lib:fref z-%data%
                                          ((f2cl-lib:int-sub
                                           (f2cl-lib:int-mul 4 n0)
                                           1))
                                          ((1 *))
                                          z-%offset%)))))))
                (cond

```

```

    (< sigma zero)
    (setf info 1)
    (go end_label)))
(setf emax zero)
(cond
  (> n0 i0)
  (setf emin
    (abs
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub
          (f2cl-lib:int-mul 4 n0)
          5))
        ((1 *))
        z-%offset%))))))
  (t
    (setf emin zero)))
(setf qmin
  (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
      3))
    ((1 *))
    z-%offset%))
(setf qmax qmin)
(f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 n0)
  (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
  (> i4 8) nil)
(tagbody
  (if
    (<=
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub i4 5))
        ((1 *))
        z-%offset%)
      zero)
    (go label100))
  (cond
    (>= qmin (* four emax))
    (setf qmin
      (min qmin
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub i4 3))
          ((1 *))
          z-%offset%))))
    (setf emax
      (max emax
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub i4 5))
          ((1 *))
          z-%offset%))))))
  (setf qmax

```

```

(max qmax
  (+
    (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub i4 7))
                    ((1 *))
                    z-%offset%)
    (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub i4 5))
                    ((1 *))
                    z-%offset%))))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub i4 5))
                    ((1 *))
                    z-%offset%))))
(setf i4 4)
label100
(setf i0 (the fixnum (truncate i4 4)))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 1))
                    ((1 *))
                    z-%offset%)
  emin)
(setf dmin
  (-
    (max zero
      (+ qmin
        (* (- two)
          (f2cl-lib:fsqrt qmin)
          (f2cl-lib:fsqrt emax))))))
(setf pp 0)
(setf nbig
  (f2cl-lib:int-mul 30
    (f2cl-lib:int-add
      (f2cl-lib:int-sub n0 i0)
      1)))
(f2cl-lib:fdo (iwhilb 1 (f2cl-lib:int-add iwhilb 1))
  ((> iwhilb nbig) nil)
  (tagbody
    (if (> i0 n0) (go label130))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9 var-10 var-11)
      (dlasq3 i0 n0 z pp dmin sigma desig qmax nfail iter ndiv
        ieee)
      (declare (ignore var-0 var-2 var-3 var-11))
      (setf n0 var-1)
      (setf dmin var-4)
      (setf sigma var-5)

```

```

(setf desig var-6)
(setf qmax var-7)
(setf nfail var-8)
(setf iter var-9)
(setf ndiv var-10))
(setf pp (f2cl-lib:int-sub 1 pp))
(cond
  ((and (= pp 0)
        (>= (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 3))
    (cond
      ((or
        (<= (f2cl-lib:fref z ((f2cl-lib:int-mul 4 n0)) ((1 *)))
          (* tol2 qmax))
        (<=
          (f2cl-lib:fref z
            ((f2cl-lib:int-add
              (f2cl-lib:int-mul 4 n0)
              (f2cl-lib:int-sub 1)))
            ((1 *)))
          (* tol2 sigma)))
        (setf splt (f2cl-lib:int-sub i0 1))
        (setf qmax
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub
              (f2cl-lib:int-mul 4 i0)
              3))
            ((1 *))
            z-%offset%))
          (setf emin
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub
                (f2cl-lib:int-mul 4 i0)
                1))
              ((1 *))
              z-%offset%))
            (setf oldemn
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-mul 4 i0))
                ((1 *))
                z-%offset%))
              (f2cl-lib:fdo (i4 (f2cl-lib:int-mul 4 i0)
                            (f2cl-lib:int-add i4 4))
                (> i4
                  (f2cl-lib:int-mul 4
                    (f2cl-lib:int-add n0
                      (f2cl-lib:int-sub
                        3))))
                  nil)
                (tagbody
                  (cond

```

```

((or
  (<= (f2cl-lib:fref z (i4) ((1 *)))
    (* tol2
      (f2cl-lib:fref z
        ((f2cl-lib:int-add i4
          (f2cl-lib:int-sub
            3))))
        ((1 *))))))

(<=
  (f2cl-lib:fref z
    ((f2cl-lib:int-add i4
      (f2cl-lib:int-sub
        1)))
    ((1 *)))
  (* tol2 sigma)))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub i4 1))
  ((1 *))
  z-%offset%)
  (- sigma))
(setf splt (the fixnum (truncate i4 4)))
(setf qmax zero)
(setf emin
  (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-add i4 3))
    ((1 *))
    z-%offset%))
(setf oldemn
  (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-add i4 4))
    ((1 *))
    z-%offset%)))
(t
  (setf qmax
    (max qmax
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add i4
          1))
        ((1 *))
        z-%offset%)))
  (setf emin
    (min emin
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub i4
          1))
        ((1 *))
        z-%offset%)))
  (setf oldemn
    (min oldemn
      (f2cl-lib:fref z-%data%

```

```

(i4)
((1 *))
z-%offset%))))))

(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub
                      (f2cl-lib:int-mul 4 n0)
                      1))
                    ((1 *))
                    z-%offset%)
      emin)
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-mul 4 n0))
                    ((1 *))
                    z-%offset%)
      oldemn)
(setf i0 (f2cl-lib:int-add splt 1))))))
(setf info 2)
(go end_label)
label130))
(setf info 3)
(go end_label)
label150
(f2cl-lib:fdo (k 2 (f2cl-lib:int-add k 1))
              (> k n) nil)
(tagbody
 (setf (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%)
       (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 k)
                                           3))
                       ((1 *))
                       z-%offset%))))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (dlasrt "D" n z iinfo)
  (declare (ignore var-0 var-1 var-2))
  (setf iinfo var-3))
(setf e zero)
(f2cl-lib:fdo (k n (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
              (> k 1) nil)
(tagbody
 (setf e (+ e (f2cl-lib:fref z-%data% (k) ((1 *)) z-%offset%))))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 1))
                    ((1 *))
                    z-%offset%)
      trace$)
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 2))
                    ((1 *))
                    z-%offset%)
      e)

```



```

      (setf (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 3))
                          ((1 *))
                          z-%offset%)
            (coerce (realpart iter) 'double-float))
      (setf (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 4))
                          ((1 *))
                          z-%offset%)
            (/ (coerce (realpart ndiv) 'double-float)
               (coerce (realpart (expt n 2)) 'double-float)))
      (setf (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add (f2cl-lib:int-mul 2 n) 5))
                          ((1 *))
                          z-%offset%)
            (/ (* hundred nfail)
               (coerce (realpart iter) 'double-float)))
      (go end_label)
end_label
      (return (values nil nil info))))))

```

dlasq3 LAPACK

— dlasq3.input —

```

)set break resume
)sys rm -f dlasq3.output
)spool dlasq3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasq3.help —

```

=====
dlasq3 examples
=====

```

```
=====
Man Page Details
=====
```

NAME

DLASQ3 - for deflation, computes a shift (TAU) and calls dqds

SYNOPSIS

```
SUBROUTINE DLASQ3( IO, NO, Z, PP, DMIN, SIGMA, DESIG, QMAX, NFAIL,
                  ITER, NDIV, IEEE )
```

```

LOGICAL      IEEE

INTEGER      IO, ITER, NO, NDIV, NFAIL, PP

DOUBLE       PRECISION DESIG, DMIN, QMAX, SIGMA

DOUBLE       PRECISION Z( * )
```

Purpose

```
=====
```

DLASQ3 checks for deflation, computes a shift (TAU) and calls dqds. In case of failure it changes shifts, and tries again until output is positive.

Arguments

```
=====
```

```

IO      (input) INTEGER
        First index.

NO      (input) INTEGER
        Last index.

Z       (input) DOUBLE PRECISION array, dimension ( 4*N )
        Z holds the qd array.

PP      (input) INTEGER
        PP=0 for ping, PP=1 for pong.

DMIN    (output) DOUBLE PRECISION
        Minimum value of d.

SIGMA   (output) DOUBLE PRECISION
        Sum of shifts used in current segment.

DESIG   (input/output) DOUBLE PRECISION
        Lower order part of SIGMA
```

QMAX (input) DOUBLE PRECISION
Maximum value of q.

NFAIL (output) INTEGER
Number of times shift was too big.

ITER (output) INTEGER
Number of iterations.

NDIV (output) INTEGER
Number of divisions.

TTYPE (output) INTEGER
Shift type.

IEEE (input) LOGICAL
Flag for IEEE or non IEEE arithmetic (passed to DLASQ5).

— dlasq3.f —

```

SUBROUTINE DLASQ3( IO, NO, Z, PP, DMIN, SIGMA, DESIG, QMAX, NFAIL,
$                ITER, NDIV, IEEE )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  May 17, 2000
*
*  .. Scalar Arguments ..
LOGICAL      IEEE
INTEGER      IO, ITER, NO, NDIV, NFAIL, PP
DOUBLE PRECISION  DESIG, DMIN, QMAX, SIGMA
*
*  .. Array Arguments ..
DOUBLE PRECISION  Z( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION  CBIAS
PARAMETER        ( CBIAS = 1.50D0 )
DOUBLE PRECISION  ZERO, QURTR, HALF, ONE, TWO, HUNDRD
PARAMETER        ( ZERO = 0.0D0, QURTR = 0.250D0, HALF = 0.5D0,
$                ONE = 1.0D0, TWO = 2.0D0, HUNDRD = 100.0D0 )
*
*  ..

```

```

*      .. Local Scalars ..
      INTEGER          IPN4, J4, NOIN, NN, TTYPE
      DOUBLE PRECISION DMIN1, DMIN2, DN, DN1, DN2, EPS, S, SAFMIN, T,
$      TAU, TEMP, TOL, TOL2
*
*      ..
*      .. External Subroutines ..
      EXTERNAL          DLASQ4, DLASQ5, DLASQ6
*
*      ..
*      .. External Function ..
      DOUBLE PRECISION  DLAMCH
      EXTERNAL          DLAMCH
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, MIN, SQRT
*
*      ..
*      .. Save statement ..
      SAVE              TTYPE
      SAVE              DMIN1, DMIN2, DN, DN1, DN2, TAU
*
*      ..
*      .. Data statement ..
      DATA             TTYPE / 0 /
      DATA             DMIN1 / ZERO /, DMIN2 / ZERO /, DN / ZERO /,
$      DN1 / ZERO /, DN2 / ZERO /, TAU / ZERO /
*
*      ..
*      .. Executable Statements ..
*
      NOIN = NO
      EPS = DLAMCH( 'Precision' )
      SAFMIN = DLAMCH( 'Safe minimum' )
      TOL = EPS*HUNDRD
      TOL2 = TOL**2
*
*      Check for deflation.
*
10  CONTINUE
*
      IF( NO.LT.IO )
$      RETURN
      IF( NO.EQ.IO )
$      GO TO 20
      NN = 4*NO + PP
      IF( NO.EQ.( IO+1 ) )
$      GO TO 40
*
*      Check whether E(N0-1) is negligible, 1 eigenvalue.
*
      IF( Z( NN-5 ).GT.TOL2*( SIGMA+Z( NN-3 ) ) .AND.
$      Z( NN-2*PP-4 ).GT.TOL2*Z( NN-7 ) )
$      GO TO 30
*

```

```

20 CONTINUE
*
      Z( 4*N0-3 ) = Z( 4*N0+PP-3 ) + SIGMA
      N0 = N0 - 1
      GO TO 10
*
*      Check whether E(N0-2) is negligible, 2 eigenvalues.
*
30 CONTINUE
*
      IF( Z( NN-9 ).GT.TOL2*SIGMA .AND.
$      Z( NN-2*PP-8 ).GT.TOL2*Z( NN-11 ) )
$      GO TO 50
*
40 CONTINUE
*
      IF( Z( NN-3 ).GT.Z( NN-7 ) ) THEN
          S = Z( NN-3 )
          Z( NN-3 ) = Z( NN-7 )
          Z( NN-7 ) = S
      END IF
      IF( Z( NN-5 ).GT.Z( NN-3 )*TOL2 ) THEN
          T = HALF*( ( Z( NN-7 )-Z( NN-3 ) )+Z( NN-5 ) )
          S = Z( NN-3 )*( Z( NN-5 ) / T )
          IF( S.LE.T ) THEN
              S = Z( NN-3 )*( Z( NN-5 ) /
$              ( T*( ONE+SQRT( ONE+S / T ) ) ) )
          ELSE
              S = Z( NN-3 )*( Z( NN-5 ) / ( T+SQRT( T )*SQRT( T+S ) ) )
          END IF
          T = Z( NN-7 ) + ( S+Z( NN-5 ) )
          Z( NN-3 ) = Z( NN-3 )*( Z( NN-7 ) / T )
          Z( NN-7 ) = T
      END IF
      Z( 4*N0-7 ) = Z( NN-7 ) + SIGMA
      Z( 4*N0-3 ) = Z( NN-3 ) + SIGMA
      N0 = N0 - 2
      GO TO 10
*
50 CONTINUE
*
*      Reverse the qd-array, if warranted.
*
      IF( DMIN.LE.ZERO .OR. NO.LT.N0IN ) THEN
          IF( CBIAS*Z( 4*I0+PP-3 ).LT.Z( 4*N0+PP-3 ) ) THEN
              IPN4 = 4*( I0+N0 )
              DO 60 J4 = 4*I0, 2*( I0+N0-1 ), 4
                  TEMP = Z( J4-3 )
                  Z( J4-3 ) = Z( IPN4-J4-3 )
                  Z( IPN4-J4-3 ) = TEMP

```

```

        TEMP = Z( J4-2 )
        Z( J4-2 ) = Z( IPN4-J4-2 )
        Z( IPN4-J4-2 ) = TEMP
        TEMP = Z( J4-1 )
        Z( J4-1 ) = Z( IPN4-J4-5 )
        Z( IPN4-J4-5 ) = TEMP
        TEMP = Z( J4 )
        Z( J4 ) = Z( IPN4-J4-4 )
        Z( IPN4-J4-4 ) = TEMP
60      CONTINUE
        IF( NO-IO.LE.4 ) THEN
            Z( 4*NO+PP-1 ) = Z( 4*IO+PP-1 )
            Z( 4*NO-PP ) = Z( 4*IO-PP )
        END IF
        DMIN2 = MIN( DMIN2, Z( 4*NO+PP-1 ) )
        Z( 4*NO+PP-1 ) = MIN( Z( 4*NO+PP-1 ), Z( 4*IO+PP-1 ),
$           Z( 4*IO+PP+3 ) )
        Z( 4*NO-PP ) = MIN( Z( 4*NO-PP ), Z( 4*IO-PP ),
$           Z( 4*IO-PP+4 ) )
        QMAX = MAX( QMAX, Z( 4*IO+PP-3 ), Z( 4*IO+PP+1 ) )
        DMIN = -ZERO
    END IF
END IF
*
70 CONTINUE
*
    IF( DMIN.LT.ZERO .OR. SAFMIN*QMAX.LT.MIN( Z( 4*NO+PP-1 ),
$       Z( 4*NO+PP-9 ), DMIN2+Z( 4*NO-PP ) ) ) THEN
*
*       Choose a shift.
*
    CALL DLASQ4( IO, NO, Z, PP, NOIN, DMIN, DMIN1, DMIN2, DN, DN1,
$       DN2, TAU, TTYPE )
*
*       Call dqds until DMIN > 0.
*
80 CONTINUE
*
    CALL DLASQ5( IO, NO, Z, PP, TAU, DMIN, DMIN1, DMIN2, DN,
$       DN1, DN2, IEEE )
*
    NDIV = NDIV + ( NO-IO+2 )
    ITER = ITER + 1
*
*       Check status.
*
    IF( DMIN.GE.ZERO .AND. DMIN1.GT.ZERO ) THEN
*
*       Success.
*

```

```

      GO TO 100
*
      ELSE IF( DMIN.LT.ZERO .AND. DMIN1.GT.ZERO .AND.
$          Z( 4*( NO-1 )-PP ).LT.TOL*( SIGMA+DN1 ) .AND.
$          ABS( DN ).LT.TOL*SIGMA ) THEN
*
*          Convergence hidden by negative DN.
*
      Z( 4*( NO-1 )-PP+2 ) = ZERO
      DMIN = ZERO
      GO TO 100
      ELSE IF( DMIN.LT.ZERO ) THEN
*
*          TAU too big. Select new TAU and try again.
*
      NFAIL = NFAIL + 1
      IF( TTYPE.LT.-22 ) THEN
*
*          Failed twice. Play it safe.
*
      TAU = ZERO
      ELSE IF( DMIN1.GT.ZERO ) THEN
*
*          Late failure. Gives excellent shift.
*
      TAU = ( TAU+DMIN )*( ONE-TWO*EPS )
      TTYPE = TTYPE - 11
      ELSE
*
*          Early failure. Divide by 4.
*
      TAU = QURTR*TAU
      TTYPE = TTYPE - 12
      END IF
      GO TO 80
      ELSE IF( DMIN.NE.DMIN ) THEN
*
*          NaN.
*
      TAU = ZERO
      GO TO 80
      ELSE
*
*          Possible underflow. Play it safe.
*
      GO TO 90
      END IF
      END IF
*
*          Risk of underflow.

```

```

*
  90 CONTINUE
      CALL DLASQ6( IO, NO, Z, PP, DMIN, DMIN1, DMIN2, DN, DN1, DN2 )
      NDIV = NDIV + ( NO-IO+2 )
      ITER = ITER + 1
      TAU = ZERO
*
  100 CONTINUE
      IF( TAU.LT.SIGMA ) THEN
          DESIG = DESIG + TAU
          T = SIGMA + DESIG
          DESIG = DESIG - ( T-SIGMA )
      ELSE
          T = SIGMA + TAU
          DESIG = SIGMA - ( T-TAU ) + DESIG
      END IF
      SIGMA = T
*
      RETURN
*
*      End of DLASQ3
*
      END

```

— LAPACK dlasq3 —

```

(let* ((cbias 1.5)
      (zero 0.0)
      (qurtr 0.25)
      (half 0.5)
      (one 1.0)
      (two 2.0)
      (hundrd 100.0))
(declare (type (double-float 1.5 1.5) cbias)
         (type (double-float 0.0 0.0) zero)
         (type (double-float 0.25 0.25) qurtr)
         (type (double-float 0.5 0.5) half)
         (type (double-float 1.0 1.0) one)
         (type (double-float 2.0 2.0) two)
         (type (double-float 100.0 100.0) hundrd))
(let ((ttype 0)
      (f2cl-lib:dmin1 zero)
      (dmin2 zero)
      (dn zero)
      (dn1 zero)
      (dn2 zero)

```



```

(tau zero))
(declare (type (double-float) tau dn2 dn1 dn dmin2 f2cl-lib:dmin1)
          (type fixnum ttype))
(defun dlasq3 (i0 n0 z pp dmin sigma desig qmax nfail iter ndiv ieee)
  (declare (type (member t nil) ieee)
            (type (double-float) qmax desig sigma dmin)
            (type (simple-array double-float (*)) z)
            (type fixnum ndiv iter nfail pp n0 i0))
  (f2cl-lib:with-multi-array-data
    ((z double-float z-%data% z-%offset%))
    (prog ((eps 0.0) (s 0.0) (safmin 0.0) (temp 0.0) (tol 0.0) (tol2 0.0)
           (ipn4 0) (j4 0) (n0in 0) (nn 0) (t$ 0.0))
      (declare (type (double-float) t$ eps s safmin temp tol tol2)
                (type fixnum ipn4 j4 n0in nn))
      (setf n0in n0)
      (setf eps (dlamch "Precision"))
      (setf safmin (dlamch "Safe minimum"))
      (setf tol (* eps hundrd))
      (setf tol2 (expt tol 2))

      (if (< n0 i0) (go end_label))
      (if (= n0 i0) (go label20))
      (setf nn (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0) pp))
      (if (= n0 (f2cl-lib:int-add i0 1)) (go label40))
      (if
        (and
          (>
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 5))
                          ((1 *))
                          z-%offset%)
            (* tol2
              (+ sigma
                (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub nn 3))
                              ((1 *))
                              z-%offset%))))))
          (>
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub
                            (f2cl-lib:int-add nn (f2cl-lib:int-mul -1 2 pp))
                            4))
                          ((1 *))
                          z-%offset%)
            (* tol2
              (f2cl-lib:fref z-%data%
                              ((f2cl-lib:int-sub nn 7))
                              ((1 *))
                              z-%offset%))))))
      (go label30))
    )
  )

```

```

label20
  (setf (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 3))
                     ((1 *))
                     z-%offset%))
    (+
      (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub
                       (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
                                         pp)
                       3))
                     ((1 *))
                     z-%offset%))
      sigma))
  (setf n0 (f2cl-lib:int-sub n0 1))
  (go label10)
label30
  (if
    (and
      (>
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub nn 9))
                       ((1 *))
                       z-%offset%))
        (* tol2 sigma))
      (>
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub
                         (f2cl-lib:int-add nn (f2cl-lib:int-mul -1 2 pp))
                         8))
                       ((1 *))
                       z-%offset%))
        (* tol2
          (f2cl-lib:fref z-%data%
                         ((f2cl-lib:int-sub nn 11))
                         ((1 *))
                         z-%offset%))))
      (go label50))
label40
  (cond
    ((>
      (f2cl-lib:fref z
                     ((f2cl-lib:int-add nn (f2cl-lib:int-sub 3))
                     ((1 *)))
      (f2cl-lib:fref z
                     ((f2cl-lib:int-add nn (f2cl-lib:int-sub 7))
                     ((1 *))))
      (setf s
        (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub nn 3))

```

```

                                ((1 *))
                                z-%offset%))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub nn 3))
                    ((1 *))
                    z-%offset%))
(f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub nn 7))
              ((1 *))
              z-%offset%))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub nn 7))
                    ((1 *))
                    z-%offset%))
                                s)))
(cond
  (>
    (f2cl-lib:fref z
                  ((f2cl-lib:int-add nn (f2cl-lib:int-sub 5)))
                  ((1 *)))

    (*
      (f2cl-lib:fref z
                    ((f2cl-lib:int-add nn (f2cl-lib:int-sub 3)))
                    ((1 *)))

      tol2))
(setf t$
  (* half
    (+
      (-
        (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub nn 7))
                      ((1 *))
                      z-%offset%))
        (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub nn 3))
                      ((1 *))
                      z-%offset%))
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub nn 5))
                    ((1 *))
                    z-%offset%))))))
(setf s
  (*
    (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub nn 3))
                  ((1 *))
                  z-%offset%)

    (/
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub nn 5))
                    ((1 *))
                    z-%offset%)
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub nn 5))
                    ((1 *))
                    z-%offset%))))))

```

```

                                ((1 *))
                                z-%offset%)
                                t$)))
(cond
  ((<= s t$)
    (setf s
      (*
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub nn 3))
          ((1 *))
          z-%offset%)
        (/
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub nn 5))
            ((1 *))
            z-%offset%)
          (* t$ (+ one (f2cl-lib:fsqrt (+ one (/ s t$))))))))))
  (t
    (setf s
      (*
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub nn 3))
          ((1 *))
          z-%offset%)
        (/
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub nn 5))
            ((1 *))
            z-%offset%)
          (+ t$
            (* (f2cl-lib:fsqrt t$)
              (f2cl-lib:fsqrt (+ t$ s))))))))))
  (setf t$
    (+
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub nn 7))
        ((1 *))
        z-%offset%)
      (+ s
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub nn 5))
          ((1 *))
          z-%offset%))))))
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub nn 3))
    ((1 *))
    z-%offset%)
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub nn 3))

```

```

((1 *))
z-%offset%)

(/
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 7))
((1 *))
z-%offset%)

t$)))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 7))
((1 *))
z-%offset%)

t$)))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 7))
((1 *))
z-%offset%)

(+
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 7))
((1 *))
z-%offset%)

sigma))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) 3))
((1 *))
z-%offset%)

(+
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 3))
((1 *))
z-%offset%)

sigma))
(setf n0 (f2cl-lib:int-sub n0 2))
(go label10)

label10
(cond
((or (<= dmin zero) (< n0 n0in))
(cond
((<
(* cbias
(f2cl-lib:fref z
((f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
PP
(f2cl-lib:int-sub 3))))
((1 *))))
(f2cl-lib:fref z
((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
PP
(f2cl-lib:int-sub 3))))

```

```

((1 *)))
(setf ipn4 (f2cl-lib:int-mul 4 (f2cl-lib:int-add i0 n0)))
(f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0)
               (f2cl-lib:int-add j4 4))
              (> j4
               (f2cl-lib:int-mul 2
                                (f2cl-lib:int-add i0
                                n0
                                (f2cl-lib:int-sub
                                1))))))
nil)
(tagbody
 (setf temp
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 3))
                        ((1 *))
                        z-%offset%))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub j4 3))
                      ((1 *))
                      z-%offset%))
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub ipn4 j4 3))
                        ((1 *))
                        z-%offset%))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub ipn4 j4 3))
                      ((1 *))
                      z-%offset%))
        temp)
 (setf temp
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 2))
                        ((1 *))
                        z-%offset%))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub j4 2))
                      ((1 *))
                      z-%offset%))
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub ipn4 j4 2))
                        ((1 *))
                        z-%offset%))
 (setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub ipn4 j4 2))
                      ((1 *))
                      z-%offset%))
        temp)
 (setf temp
        (f2cl-lib:fref z-%data%

```

```

((f2cl-lib:int-sub j4 1))
((1 *))
z-%offset%)
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub j4 1))
((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 j4 5))
((1 *))
z-%offset%))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 j4 5))
((1 *))
z-%offset%)
temp)
(setf temp
(f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))
(setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 j4 4))
((1 *))
z-%offset%))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub ipn4 j4 4))
((1 *))
z-%offset%)
temp)))
(cond
((<= (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 4)
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add
(f2cl-lib:int-mul 4 n0)
pp)
1))
((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-add
(f2cl-lib:int-mul 4 i0)
pp)
1))
((1 *))
z-%offset%))
(setf (f2cl-lib:fref z-%data%
((f2cl-lib:int-sub
(f2cl-lib:int-mul 4 n0)
pp))

```

```

((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-mul 4 i0)
    pp))
  ((1 *))
  z-%offset%)))
(setf dmin2
  (min dmin2
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub
        (f2cl-lib:int-add
          (f2cl-lib:int-mul 4 n0)
          pp)
        1))
      ((1 *))
      z-%offset%)))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub
    (f2cl-lib:int-add
      (f2cl-lib:int-mul 4 n0)
      pp)
    1))
  ((1 *))
  z-%offset%)
  (min
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub
        (f2cl-lib:int-add
          (f2cl-lib:int-mul 4 n0)
          pp)
        1))
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub
        (f2cl-lib:int-add
          (f2cl-lib:int-mul 4 i0)
          pp)
        1))
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add
        (f2cl-lib:int-mul 4 i0)
        pp)
      3))
      ((1 *))
      z-%offset%)))

```



```

(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0)
                                         pp))
                    ((1 *))
                    z-%offset%))
(min
 (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub
                  (f2cl-lib:int-mul 4 n0)
                  pp))
                ((1 *))
                z-%offset%))
(f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub
                  (f2cl-lib:int-mul 4 i0)
                  pp))
                ((1 *))
                z-%offset%))
(f2cl-lib:fref z-%data%
                ((f2cl-lib:int-add
                  (f2cl-lib:int-sub
                    (f2cl-lib:int-mul 4 i0)
                    pp)
                  4))
                ((1 *))
                z-%offset%)))
(setf qmax
      (max qmax
            (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub
                              (f2cl-lib:int-add
                                (f2cl-lib:int-mul 4 i0)
                                pp)
                              3))
                            ((1 *))
                            z-%offset%))
            (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-add
                              (f2cl-lib:int-mul 4 i0)
                              pp
                              1))
                            ((1 *))
                            z-%offset%))))
(setf dmin (- zero))))))
(cond
 ((or (< dmin zero)
      (< (* safmin qmax)
          (min
            (f2cl-lib:fref z
                          ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)

```

```

                                                    pp
                                                    (f2cl-lib:int-sub 1)))
                    ((1 *)))
(f2cl-lib:fref z
  ((f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
    pp
    (f2cl-lib:int-sub 9)))
  ((1 *)))
(+ dmin2
  (f2cl-lib:fref z
    ((f2cl-lib:int-add
      (f2cl-lib:int-mul 4 n0)
      (f2cl-lib:int-sub pp)))
    ((1 *))))))
(tagbody
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11 var-12)
    (dlasq4 i0 n0 z pp n0in dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2
     tau ttype)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9 var-10))
    (setf tau var-11)
    (setf ttype var-12))
  label80
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11)
    (dlasq5 i0 n0 z pp tau dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2
     ieee)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6 var-11))
    (setf dmin var-5)
    (setf dmin2 var-7)
    (setf dn var-8)
    (setf dn1 var-9)
    (setf dn2 var-10))
  (setf ndiv
    (f2cl-lib:int-add ndiv
      (f2cl-lib:int-add
        (f2cl-lib:int-sub n0 i0)
        2)))
  (setf iter (f2cl-lib:int-add iter 1))
  (cond
    ((and (>= dmin zero) (> f2cl-lib:dmin1 zero))
     (go label100))
    ((and (< dmin zero)
      (> f2cl-lib:dmin1 zero)
      (<
        (f2cl-lib:fref z
          ((f2cl-lib:int-add

```

```

                                (f2cl-lib:int-mul 4
                                (f2cl-lib:int-add n0
                                (f2cl-lib:int-sub
                                1)))
                                (f2cl-lib:int-sub pp)))
                                ((1 *)))
                                (* tol (+ sigma dn1)))
                                (< (abs dn) (* tol sigma)))
(setf (f2cl-lib:fref z-%data%
                                ((f2cl-lib:int-add
                                (f2cl-lib:int-sub
                                (f2cl-lib:int-mul 4
                                (f2cl-lib:int-sub
                                n0
                                1))
                                pp)
                                2))
                                ((1 *))
                                z-%offset%))
                                zero)
(setf dmin zero)
(go label100))
(< dmin zero)
(setf nfail (f2cl-lib:int-add nfail 1))
(cond
  (< ttype (f2cl-lib:int-sub 22))
  (setf tau zero))
(> f2cl-lib:dmin1 zero)
(setf tau (* (+ tau dmin) (- one (* two eps))))
(setf ttype (f2cl-lib:int-sub ttype 11)))
(t
  (setf tau (* qurtr tau))
  (setf ttype (f2cl-lib:int-sub ttype 12))))
(go label80))
(/= dmin dmin)
(setf tau zero)
(go label80))
(t
  (go label90))))))
label90
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
  (dlasq6 i0 n0 z pp dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2)
  (declare (ignore var-0 var-1 var-2 var-3 var-5))
  (setf dmin var-4)
  (setf dmin2 var-6)
  (setf dn var-7)
  (setf dn1 var-8)
  (setf dn2 var-9))
(setf ndiv

```

```

(f2cl-lib:int-add ndiv
  (f2cl-lib:int-add (f2cl-lib:int-sub n0 i0)
    2)))

(setf iter (f2cl-lib:int-add iter 1))
(setf tau zero)
label100
(cond
  ((< tau sigma)
    (setf desig (+ desig tau))
    (setf t$ (+ sigma desig))
    (setf desig (- desig (- t$ sigma))))
  (t
    (setf t$ (+ sigma tau))
    (setf desig (+ (- sigma (- t$ tau)) desig))))
(setf sigma t$)
end_label
(return
  (values nil
    n0
    nil
    nil
    dmin
    sigma
    desig
    qmax
    nfail
    iter
    ndiv
    nil))))))

```

dlasq4 LAPACK

— dlasq4.input —

```

)set break resume
)sys rm -f dlasq4.output
)spool dlasq4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasq4.help —

```
=====
dlasq4 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASQ4 - an approximation TAU to the smallest eigenvalue using values of d from the previous transform

SYNOPSIS

```
SUBROUTINE DLASQ4( IO, NO, Z, PP, NOIN, DMIN, DMIN1, DMIN2, DN, DN1,
                  DN2, TAU, TTYPE )
```

```
      INTEGER      IO, NO, NOIN, PP, TTYPE
```

```
      DOUBLE      PRECISION DMIN, DMIN1, DMIN2, DN, DN1, DN2, TAU
```

```
      DOUBLE      PRECISION Z( * )
```

Purpose

```
=====
```

DLASQ4 computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform.

IO (input) INTEGER
First index.

NO (input) INTEGER
Last index.

Z (input) DOUBLE PRECISION array, dimension (4*N)
Z holds the qd array.

PP (input) INTEGER
PP=0 for ping, PP=1 for pong.

NOIN (input) INTEGER
The value of NO at start of EIGTEST.

DMIN (input) DOUBLE PRECISION
Minimum value of d.

DMIN1 (input) DOUBLE PRECISION
Minimum value of d, excluding D(NO).

DMIN2 (input) DOUBLE PRECISION
Minimum value of d, excluding D(NO) and D(NO-1).

DN (input) DOUBLE PRECISION
d(N)

DN1 (input) DOUBLE PRECISION
d(N-1)

DN2 (input) DOUBLE PRECISION
d(N-2)

TAU (output) DOUBLE PRECISION
This is the shift.

TTYTYPE (output) INTEGER
Shift type.

Further Details

=====

CNST1 = 9/16

— dlasq4.f —

```

      SUBROUTINE DLASQ4( IO, NO, Z, PP, NOIN, DMIN, DMIN1, DMIN2, DN,
$      DN1, DN2, TAU, TTYPE )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1999
*
*  .. Scalar Arguments ..
*  INTEGER          IO, NO, NOIN, PP, TTYPE
*  DOUBLE PRECISION DMIN, DMIN1, DMIN2, DN, DN1, DN2, TAU
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION Z( * )
*
*  ..
*
*  =====
*

```

```

*      .. Parameters ..
DOUBLE PRECISION   CNST1, CNST2, CNST3
PARAMETER           ( CNST1 = 0.5630D0, CNST2 = 1.010D0,
$                   CNST3 = 1.050D0 )
DOUBLE PRECISION   QURTR, THIRD, HALF, ZERO, ONE, TWO, HUNDRD
PARAMETER           ( QURTR = 0.250D0, THIRD = 0.3330D0,
$                   HALF = 0.50D0, ZERO = 0.0D0, ONE = 1.0D0,
$                   TWO = 2.0D0, HUNDRD = 100.0D0 )

*      ..
*      .. Local Scalars ..
INTEGER             I4, NN, NP
DOUBLE PRECISION    A2, B1, B2, G, GAM, GAP1, GAP2, S

*      ..
*      .. Intrinsic Functions ..
INTRINSIC           MAX, MIN, SQRT

*      ..
*      .. Save statement ..
SAVE                G

*      ..
*      .. Data statement ..
DATA                G / ZERO /

*      ..
*      .. Executable Statements ..

*      A negative DMIN forces the shift to take that absolute value
*      TTYPE records the type of shift.
*
IF( DMIN.LE.ZERO ) THEN
    TAU = -DMIN
    TTYPE = -1
    RETURN
END IF

*
NN = 4*NO + PP
IF( NOIN.EQ.NO ) THEN

*
*      No eigenvalues deflated.
*
IF( DMIN.EQ.DN .OR. DMIN.EQ.DN1 ) THEN

*
    B1 = SQRT( Z( NN-3 ) ) * SQRT( Z( NN-5 ) )
    B2 = SQRT( Z( NN-7 ) ) * SQRT( Z( NN-9 ) )
    A2 = Z( NN-7 ) + Z( NN-5 )

*
*      Cases 2 and 3.
*
IF( DMIN.EQ.DN .AND. DMIN1.EQ.DN1 ) THEN
    GAP2 = DMIN2 - A2 - DMIN2*QURTR
    IF( GAP2.GT.ZERO .AND. GAP2.GT.B2 ) THEN
        GAP1 = A2 - DN - ( B2 / GAP2 ) * B2
    
```

```

ELSE
    GAP1 = A2 - DN - ( B1+B2 )
END IF
IF( GAP1.GT.ZERO .AND. GAP1.GT.B1 ) THEN
    S = MAX( DN-( B1 / GAP1 )*B1, HALF*DMIN )
    TTYPE = -2
ELSE
    S = ZERO
    IF( DN.GT.B1 )
$       S = DN - B1
    IF( A2.GT.( B1+B2 ) )
$       S = MIN( S, A2-( B1+B2 ) )
    S = MAX( S, THIRD*DMIN )
    TTYPE = -3
END IF
ELSE
*
*       Case 4.
*
    TTYPE = -4
    S = QURTR*DMIN
    IF( DMIN.EQ.DN ) THEN
        GAM = DN
        A2 = ZERO
        IF( Z( NN-5 ) .GT. Z( NN-7 ) )
$           RETURN
        B2 = Z( NN-5 ) / Z( NN-7 )
        NP = NN - 9
    ELSE
        NP = NN - 2*PP
        B2 = Z( NP-2 )
        GAM = DN1
        IF( Z( NP-4 ) .GT. Z( NP-2 ) )
$           RETURN
        A2 = Z( NP-4 ) / Z( NP-2 )
        IF( Z( NN-9 ) .GT. Z( NN-11 ) )
$           RETURN
        B2 = Z( NN-9 ) / Z( NN-11 )
        NP = NN - 13
    END IF
*
*       Approximate contribution to norm squared from I < NN-1.
*
    A2 = A2 + B2
    DO 10 I4 = NP, 4*I0 - 1 + PP, -4
        IF( B2.EQ.ZERO )
$            GO TO 20
        B1 = B2
        IF( Z( I4 ) .GT. Z( I4-2 ) )
$            RETURN

```



```

        B2 = B2*( Z( I4 ) / Z( I4-2 ) )
        A2 = A2 + B2
        IF( HUNDRD*MAX( B2, B1 ).LT.A2 .OR. CNST1.LT.A2 )
$           GO TO 20
10        CONTINUE
20        CONTINUE
        A2 = CNST3*A2
*
*           Rayleigh quotient residual bound.
*
        IF( A2.LT.CNST1 )
$           S = GAM*( ONE-SQRT( A2 ) ) / ( ONE+A2 )
        END IF
        ELSE IF( DMIN.EQ.DN2 ) THEN
*
*           Case 5.
*
        TTYPE = -5
        S = QURTR*DMIN
*
*           Compute contribution to norm squared from I > NN-2.
*
        NP = NN - 2*PP
        B1 = Z( NP-2 )
        B2 = Z( NP-6 )
        GAM = DN2
        IF( Z( NP-8 ).GT.B2 .OR. Z( NP-4 ).GT.B1 )
$           RETURN
        A2 = ( Z( NP-8 ) / B2 )*( ONE+Z( NP-4 ) / B1 )
*
*           Approximate contribution to norm squared from I < NN-2.
*
        IF( NO-I0.GT.2 ) THEN
            B2 = Z( NN-13 ) / Z( NN-15 )
            A2 = A2 + B2
            DO 30 I4 = NN - 17, 4*I0 - 1 + PP, -4
                IF( B2.EQ.ZERO )
$                    GO TO 40
                    B1 = B2
                    IF( Z( I4 ) .GT. Z( I4-2 ) )
$                        RETURN
                    B2 = B2*( Z( I4 ) / Z( I4-2 ) )
                    A2 = A2 + B2
                    IF( HUNDRD*MAX( B2, B1 ).LT.A2 .OR. CNST1.LT.A2 )
$                        GO TO 40
30                CONTINUE
40                CONTINUE
                A2 = CNST3*A2
            END IF
*

```

```

        IF( A2.LT.CNST1 )
$      S = GAM*( ONE-SQRT( A2 ) ) / ( ONE+A2 )
    ELSE
*
*      Case 6, no information to guide us.
*
        IF( TTYPE.EQ.-6 ) THEN
            G = G + THIRD*( ONE-G )
        ELSE IF( TTYPE.EQ.-18 ) THEN
            G = QURTR*THIRD
        ELSE
            G = QURTR
        END IF
        S = G*DMIN
        TTYPE = -6
    END IF
*
    ELSE IF( NOIN.EQ.( NO+1 ) ) THEN
*
*      One eigenvalue just deflated. Use DMIN1, DN1 for DMIN and DN.
*
        IF( DMIN1.EQ.DN1 .AND. DMIN2.EQ.DN2 ) THEN
*
*      Cases 7 and 8.
*
            TTYPE = -7
            S = THIRD*DMIN1
            IF( Z( NN-5 ).GT.Z( NN-7 ) )
$              RETURN
            B1 = Z( NN-5 ) / Z( NN-7 )
            B2 = B1
            IF( B2.EQ.ZERO )
$              GO TO 60
            DO 50 I4 = 4*NO - 9 + PP, 4*IO - 1 + PP, -4
                A2 = B1
                IF( Z( I4 ).GT.Z( I4-2 ) )
$                  RETURN
                B1 = B1*( Z( I4 ) / Z( I4-2 ) )
                B2 = B2 + B1
                IF( HUNDRD*MAX( B1, A2 ).LT.B2 )
$                  GO TO 60
50          CONTINUE
60          CONTINUE
            B2 = SQRT( CNST3*B2 )
            A2 = DMIN1 / ( ONE+B2**2 )
            GAP2 = HALF*DMIN2 - A2
            IF( GAP2.GT.ZERO .AND. GAP2.GT.B2*A2 ) THEN
                S = MAX( S, A2*( ONE-CNST2*A2*( B2 / GAP2 )*B2 ) )
            ELSE
                S = MAX( S, A2*( ONE-CNST2*B2 ) )

```

```

        TTYPE = -8
    END IF
ELSE
*
*       Case 9.
*
        S = QURTR*DMIN1
        IF( DMIN1.EQ.DN1 )
$           S = HALF*DMIN1
        TTYPE = -9
    END IF
*
ELSE IF( NOIN.EQ.( NO+2 ) ) THEN
*
*       Two eigenvalues deflated. Use DMIN2, DN2 for DMIN and DN.
*
*       Cases 10 and 11.
*
        IF( DMIN2.EQ.DN2 .AND. TWO*Z( NN-5 ).LT.Z( NN-7 ) ) THEN
            TTYPE = -10
            S = THIRD*DMIN2
            IF( Z( NN-5 ).GT.Z( NN-7 ) )
$                RETURN
            B1 = Z( NN-5 ) / Z( NN-7 )
            B2 = B1
            IF( B2.EQ.ZERO )
$                GO TO 80
            DO 70 I4 = 4*NO - 9 + PP, 4*IO - 1 + PP, -4
                IF( Z( I4 ).GT.Z( I4-2 ) )
$                    RETURN
                B1 = B1*( Z( I4 ) / Z( I4-2 ) )
                B2 = B2 + B1
                IF( HUNDRD*B1.LT.B2 )
$                    GO TO 80
70          CONTINUE
80          CONTINUE
            B2 = SQRT( CNST3*B2 )
            A2 = DMIN2 / ( ONE+B2**2 )
            GAP2 = Z( NN-7 ) + Z( NN-9 ) -
$                SQRT( Z( NN-11 ) )*SQRT( Z( NN-9 ) ) - A2
            IF( GAP2.GT.ZERO .AND. GAP2.GT.B2*A2 ) THEN
                S = MAX( S, A2*( ONE-CNST2*A2*( B2 / GAP2 )*B2 ) )
            ELSE
                S = MAX( S, A2*( ONE-CNST2*B2 ) )
            END IF
        ELSE
            S = QURTR*DMIN2
            TTYPE = -11
        END IF
    ELSE IF( NOIN.GT.( NO+2 ) ) THEN

```

```

*
*      Case 12, more than two eigenvalues deflated. No information.
*
      S = ZERO
      TTYPE = -12
      END IF
*
      TAU = S
      RETURN
*
*      End of DLASQ4
*
      END

```

— LAPACK dlasq4 —

```

(let* ((cnst1 0.563)
      (cnst2 1.01)
      (cnst3 1.05)
      (qurtr 0.25)
      (third$ 0.333)
      (half 0.5)
      (zero 0.0)
      (one 1.0)
      (two 2.0)
      (hundrd 100.0))
(declare (type (double-float 0.563 0.563) cnst1)
         (type (double-float 1.01 1.01) cnst2)
         (type (double-float 1.05 1.05) cnst3)
         (type (double-float 0.25 0.25) qurtr)
         (type (double-float 0.333 0.333) third$)
         (type (double-float 0.5 0.5) half)
         (type (double-float 0.0 0.0) zero)
         (type (double-float 1.0 1.0) one)
         (type (double-float 2.0 2.0) two)
         (type (double-float 100.0 100.0) hundrd))
(let ((g zero))
(declare (type (double-float) g))
(defun dlasq4
  (i0 n0 z pp n0in dmin f2cl-lib:dmin1 dmin2 dn dn1 dn2 tau ttype)
  (declare (type (double-float) tau dn2 dn1 dn dmin2 f2cl-lib:dmin1 dmin)
           (type (simple-array double-float (*)) z)
           (type fixnum ttype n0in pp n0 i0))
  (f2cl-lib:with-multi-array-data
    ((z double-float z-%data% z-%offset%))
    (prog ((a2 0.0) (b1 0.0) (b2 0.0) (gam 0.0) (gap1 0.0) (gap2 0.0)

```

```

(s 0.0) (i4 0) (nn 0) (np 0) (sqrt$ 0.0f0))
(declare (type (single-float) sqrt$)
  (type (double-float) a2 b1 b2 gam gap1 gap2 s)
  (type fixnum i4 nn np))
(cond
  ((<= dmin zero)
    (setf tau (- dmin))
    (setf ttype -1)
    (go end_label)))
(setf nn (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0) pp))
(cond
  ((= n0in n0)
    (cond
      ((or (= dmin dn) (= dmin dn1))
        (setf b1
          (*
            (f2cl-lib:fsqrt
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub nn 3))
                ((1 *))
                z-%offset%))
            (f2cl-lib:fsqrt
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub nn 5))
                ((1 *))
                z-%offset%))))))
        (setf b2
          (*
            (f2cl-lib:fsqrt
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub nn 7))
                ((1 *))
                z-%offset%))
            (f2cl-lib:fsqrt
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub nn 9))
                ((1 *))
                z-%offset%))))))
        (setf a2
          (+
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub nn 7))
              ((1 *))
              z-%offset%)
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub nn 5))
              ((1 *))
              z-%offset%))))))
      ((and (= dmin dn) (= f2cl-lib:dmin1 dn1))

```

```

(setf gap2 (- dmin2 a2 (* dmin2 qurtr)))
(cond
  ((and (> gap2 zero) (> gap2 b2))
    (setf gap1 (- a2 dn (* (/ b2 gap2) b2))))
  (t
    (setf gap1 (- a2 dn (+ b1 b2)))))
(cond
  ((and (> gap1 zero) (> gap1 b1))
    (setf s (max (- dn (* (/ b1 gap1) b1)) (* half dmin)))
    (setf ttype -2))
  (t
    (setf s zero)
    (if (> dn b1) (setf s (- dn b1)))
    (if (> a2 (+ b1 b2)) (setf s (min s (- a2 (+ b1 b2)))))
    (setf s (max s (* third$ dmin)))
    (setf ttype -3))))
(t
  (tagbody
    (setf ttype -4)
    (setf s (* qurtr dmin))
    (cond
      ((= dmin dn)
        (setf gam dn)
        (setf a2 zero)
        (if
          (>
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 5))
                          ((1 *))
                          z-%offset%)
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 7))
                          ((1 *))
                          z-%offset%))
          (go end_label))
        (setf b2
          (/
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 5))
                          ((1 *))
                          z-%offset%)
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub nn 7))
                          ((1 *))
                          z-%offset%)))
        (setf np (f2cl-lib:int-sub nn 9)))
      (t
        (setf np (f2cl-lib:int-sub nn (f2cl-lib:int-mul 2 pp)))
        (setf b2
          (f2cl-lib:fref z-%data%

```

```

((f2cl-lib:int-sub np 2))
((1 *))
z-%offset%))

(setf gam dn1)
(if
  (>
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub np 4))
      ((1 *))
      z-%offset%))
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub np 2))
      ((1 *))
      z-%offset%))
    (go end_label))
(setf a2
  (/
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub np 4))
      ((1 *))
      z-%offset%))
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub np 2))
      ((1 *))
      z-%offset%)))
(if
  (>
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 9))
      ((1 *))
      z-%offset%))
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 11))
      ((1 *))
      z-%offset%))
    (go end_label))
(setf b2
  (/
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 9))
      ((1 *))
      z-%offset%))
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 11))
      ((1 *))
      z-%offset%)))
(setf np (f2cl-lib:int-sub nn 13)))
(setf a2 (+ a2 b2))
(f2cl-lib:fdo (i4 np
  (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))

```

```

                                (> i4
                                (f2cl-lib:int-add
                                (f2cl-lib:int-mul 4 i0)
                                (f2cl-lib:int-sub 1)
                                pp))
                                nil)
(tagbody
  (if (= b2 zero) (go label20))
  (setf b1 b2)
  (if
    (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub i4 2))
                    ((1 *))
                    z-%offset%))
    (go end_label))
  (setf b2
    (* b2
      (/
        (f2cl-lib:fref z-%data%
                      (i4)
                      ((1 *))
                      z-%offset%)
        (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub i4 2))
                      ((1 *))
                      z-%offset%))))))
  (setf a2 (+ a2 b2))
  (if (or (< (* hundred (max b2 b1)) a2) (< cnst1 a2))
    (go label20)))

label20

  (setf a2 (* cnst3 a2))
  (if (< a2 cnst1)
    (setf s
      (/ (* gam (- one (f2cl-lib:fsqrt a2)))
        (+ one a2))))))

(= dmin dn2)
(setf ttype -5)
(setf s (* qurtr dmin))
(setf np (f2cl-lib:int-sub nn (f2cl-lib:int-mul 2 pp)))
(setf b1
  (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub np 2))
                ((1 *))
                z-%offset%))

(setf b2
  (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub np 6))
                ((1 *))
                z-%offset%))

```



```

(setf gam dn2)
(if
  (or
    (>
      (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub np 8))
                     ((1 *))
                     z-%offset%)

      b2)
    (>
      (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub np 4))
                     ((1 *))
                     z-%offset%)

      b1))
  (go end_label))
(setf a2
  (*
    (/
      (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub np 8))
                     ((1 *))
                     z-%offset%)

      b2)
    (+ one
      (/
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub np 4))
                        ((1 *))
                        z-%offset%)

        b1))))))
(cond
  ((> (f2cl-lib:int-add n0 (f2cl-lib:int-sub i0)) 2)
   (tagbody
    (setf b2
      (/
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub nn 13))
                        ((1 *))
                        z-%offset%)

        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub nn 15))
                        ((1 *))
                        z-%offset%)))

    (setf a2 (+ a2 b2))
    (f2cl-lib:fdo (i4
      (f2cl-lib:int-add nn (f2cl-lib:int-sub 17))
      (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
      (> i4
        (f2cl-lib:int-add

```

```

                                (f2cl-lib:int-mul 4 i0)
                                (f2cl-lib:int-sub 1)
                                pp))
                                nil)
(tagbody
  (if (= b2 zero) (go label40))
  (setf b1 b2)
  (if
    (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub i4 2))
                    ((1 *))
                    z-%offset%)))
    (go end_label))
  (setf b2
    (* b2
      (/
        (f2cl-lib:fref z-%data%
                      (i4)
                      ((1 *))
                      z-%offset%)
        (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub i4 2))
                      ((1 *))
                      z-%offset%))))))
  (setf a2 (+ a2 b2))
  (if (or (< (* hundred (max b2 b1)) a2) (< cnst1 a2))
    (go label40))))

label40
  (setf a2 (* cnst3 a2))))
(if (< a2 cnst1)
  (setf s
    (/ (* gam (- one (f2cl-lib:fsqrt a2)))
      (+ one a2))))
(t
  (cond
    ((= ttype (f2cl-lib:int-sub 6))
     (setf g (+ g (* third$ (- one g)))))
    ((= ttype (f2cl-lib:int-sub 18))
     (setf g (* qurtr third$)))
    (t
     (setf g qurtr)))
  (setf s (* g dmin))
  (setf ttype -6))))
(= n0in (f2cl-lib:int-add n0 1))
(cond
  ((and (= f2cl-lib:dmin1 dn1) (= dmin2 dn2))
   (tagbody
     (setf ttype -7)
     (setf s (* third$ f2cl-lib:dmin1))

```

```
(if
(>
  (f2c1-lib:fref z-%data%
    ((f2c1-lib:int-sub nn 5))
    ((1 *))
    z-%offset%)
  (f2c1-lib:fref z-%data%
    ((f2c1-lib:int-sub nn 7))
    ((1 *))
    z-%offset%))
(go end_label))
(setf b1
  (/
    (f2c1-lib:fref z-%data%
      ((f2c1-lib:int-sub nn 5))
      ((1 *))
      z-%offset%)
    (f2c1-lib:fref z-%data%
      ((f2c1-lib:int-sub nn 7))
      ((1 *))
      z-%offset%)))
(setf b2 b1)
(if (= b2 zero) (go label60))
(f2c1-lib:fdo (i4
  (f2c1-lib:int-add (f2c1-lib:int-mul 4 n0)
    (f2c1-lib:int-sub 9)
    pp)
  (f2c1-lib:int-add i4 (f2c1-lib:int-sub 4)))
(> i4
  (f2c1-lib:int-add (f2c1-lib:int-mul 4 i0)
    (f2c1-lib:int-sub 1)
    pp))
  nil)
(tagbody
  (setf a2 b1)
  (if
    (> (f2c1-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
      (f2c1-lib:fref z-%data%
        ((f2c1-lib:int-sub i4 2))
        ((1 *))
        z-%offset%))
    (go end_label))
  (setf b1
    (* b1
      (/
        (f2c1-lib:fref z-%data%
          (i4)
          ((1 *))
          z-%offset%)
        (f2c1-lib:fref z-%data%
```

```

((f2cl-lib:int-sub i4 2))
((1 *))
z-%offset%)))

(setf b2 (+ b2 b1))
(if (< (* hundrd (max b1 a2)) b2) (go label60))))

label60

(setf b2 (f2cl-lib:fsqrt (* cnst3 b2)))
(setf a2 (/ f2cl-lib:dmin1 (+ one (expt b2 2))))
(setf gap2 (- (* half dmin2) a2))
(cond
  ((and (> gap2 zero) (> gap2 (* b2 a2)))
    (setf s
      (max s
        (* a2
          (+ one (* (- cnst2) a2 (/ b2 gap2) b2))))))
    (t
      (setf s (max s (* a2 (- one (* cnst2 b2)))))
      (setf ttype -8))))))
(t
  (setf s (* qurtr f2cl-lib:dmin1))
  (if (= f2cl-lib:dmin1 dn1) (setf s (* half f2cl-lib:dmin1)))
  (setf ttype -9))))
(= n0in (f2cl-lib:int-add n0 2))
(cond
  ((and (= dmin2 dn2)
    (<
      (* two
        (f2cl-lib:fref z
          ((f2cl-lib:int-add nn
            (f2cl-lib:int-sub
              5)))
          ((1 *))))
        (f2cl-lib:fref z
          ((f2cl-lib:int-add nn
            (f2cl-lib:int-sub 7)))
          ((1 *))))))
    (tagbody
      (setf ttype -10)
      (setf s (* third$ dmin2))
      (if
        (>
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub nn 5))
            ((1 *))
            z-%offset%)
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub nn 7))
            ((1 *))
            z-%offset%))
        (go end_label))

```

```

(setf b1
  (/
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 5))
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub nn 7))
      ((1 *))
      z-%offset%)))
(setf b2 b1)
(if (= b2 zero) (go label80))
(f2cl-lib:fd0 i4
  (f2cl-lib:int-add (f2cl-lib:int-mul 4 n0)
    (f2cl-lib:int-sub 9)
    pp)
  (f2cl-lib:int-add i4 (f2cl-lib:int-sub 4)))
(> i4
  (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0)
    (f2cl-lib:int-sub 1)
    pp))
  nil)
(tagbody
  (if
    (> (f2cl-lib:fref z-%data% (i4) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub i4 2))
        ((1 *))
        z-%offset%)))
    (go end_label))
  (setf b1
    (* b1
      (/
        (f2cl-lib:fref z-%data%
          (i4)
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub i4 2))
          ((1 *))
          z-%offset%))))))
  (setf b2 (+ b2 b1))
  (if (< (* hundrd b1) b2) (go label80))))

label80
(setf b2 (f2cl-lib:fsqrt (* cnst3 b2)))
(setf a2 (/ dmin2 (+ one (expt b2 2))))
(setf gap2
  (-
    (+
      (f2cl-lib:fref z-%data%

```

```

((f2cl-lib:int-sub nn 7))
((1 *))
z-%offset%)
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 9))
((1 *))
z-%offset%))
(*
(f2cl-lib:fsqrt
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 11))
((1 *))
z-%offset%))
(f2cl-lib:fsqrt
(f2cl-lib:fref z-%data%
((f2cl-lib:int-sub nn 9))
((1 *))
z-%offset%)))
a2))
(cond
((and (> gap2 zero) (> gap2 (* b2 a2)))
(setf s
(max s
(* a2
(+ one (* (- cnst2) a2 (/ b2 gap2) b2))))))
(t
(setf s (max s (* a2 (- one (* cnst2 b2)))))))
(t
(setf s (* qurtr dmin2))
(setf ttype -11)))
(> n0in (f2cl-lib:int-add n0 2))
(setf s zero)
(setf ttype -12)))
(setf tau s)
end_label
(return
(values nil nil nil nil nil nil nil nil nil tau ttype))))))

```

dlasq5 LAPACK

— dlasq5.input —

```

)set break resume
)sys rm -f dlasq5.output

```

```
)spool dlasq5.output
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

— dlasq5.help —

```
=====
dlasq5 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASQ5 - one dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines

SYNOPSIS

```
SUBROUTINE DLASQ5( IO, NO, Z, PP, TAU, DMIN, DMIN1, DMIN2, DN, DNM1,
                  DNM2, IEEE )
```

LOGICAL IEEE

INTEGER IO, NO, PP

DOUBLE PRECISION DMIN, DMIN1, DMIN2, DN, DNM1, DNM2, TAU

DOUBLE PRECISION Z(*)

Purpose

```
=====
```

DLASQ5 computes one dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines.

Arguments

```
=====
```

IO (input) INTEGER
First index.

NO (input) INTEGER

Last index.

Z (input) DOUBLE PRECISION array, dimension (4*N)
Z holds the qd array. EMIN is stored in Z(4*N0) to avoid
an extra argument.

PP (input) INTEGER
PP=0 for ping, PP=1 for pong.

TAU (input) DOUBLE PRECISION
This is the shift.

DMIN (output) DOUBLE PRECISION
Minimum value of d.

DMIN1 (output) DOUBLE PRECISION
Minimum value of d, excluding D(NO).

DMIN2 (output) DOUBLE PRECISION
Minimum value of d, excluding D(NO) and D(NO-1).

DN (output) DOUBLE PRECISION
d(NO), the last value of d.

DNM1 (output) DOUBLE PRECISION
d(NO-1).

DNM2 (output) DOUBLE PRECISION
d(NO-2).

IEEE (input) LOGICAL
Flag for IEEE or non IEEE arithmetic.

— dlasq5.f —

```

SUBROUTINE DLASQ5( IO, NO, Z, PP, TAU, DMIN, DMIN1, DMIN2, DN,
$                DNM1, DNM2, IEEE )
*
* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   May 17, 2000
*
* .. Scalar Arguments ..
LOGICAL          IEEE
INTEGER          IO, NO, PP

```



```

      DOUBLE PRECISION  DMIN, DMIN1, DMIN2, DN, DNM1, DNM2, TAU
*
*  ..
*  .. Array Arguments ..
      DOUBLE PRECISION  Z( * )
*
*  ..
*
*  =====
*
*  .. Parameter ..
      DOUBLE PRECISION  ZERO
      PARAMETER          ( ZERO = 0.0D0 )
*
*  ..
*  .. Local Scalars ..
      INTEGER            J4, J4P2
      DOUBLE PRECISION  D, EMIN, TEMP
*
*  ..
*  .. Intrinsic Functions ..
      INTRINSIC          MIN
*
*  ..
*  .. Executable Statements ..
*
      IF( ( NO-IO-1 ).LE.0 )
$      RETURN
*
      J4 = 4*IO + PP - 3
      EMIN = Z( J4+4 )
      D = Z( J4 ) - TAU
      DMIN = D
      DMIN1 = -Z( J4 )
*
      IF( IEEE ) THEN
*
*       Code for IEEE arithmetic.
*
      IF( PP.EQ.0 ) THEN
          DO 10 J4 = 4*IO, 4*( NO-3 ), 4
              Z( J4-2 ) = D + Z( J4-1 )
              TEMP = Z( J4+1 ) / Z( J4-2 )
              D = D*TEMP - TAU
              DMIN = MIN( DMIN, D )
              Z( J4 ) = Z( J4-1 )*TEMP
              EMIN = MIN( Z( J4 ), EMIN )
10          CONTINUE
      ELSE
          DO 20 J4 = 4*IO, 4*( NO-3 ), 4
              Z( J4-3 ) = D + Z( J4 )
              TEMP = Z( J4+2 ) / Z( J4-3 )
              D = D*TEMP - TAU
              DMIN = MIN( DMIN, D )
              Z( J4-1 ) = Z( J4 )*TEMP

```

```

                EMIN = MIN( Z( J4-1 ), EMIN )
20          CONTINUE
          END IF
*
*          Unroll last two steps.
*
          DNM2 = D
          DMIN2 = DMIN
          J4 = 4*( NO-2 ) - PP
          J4P2 = J4 + 2*PP - 1
          Z( J4-2 ) = DNM2 + Z( J4P2 )
          Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
          DNM1 = Z( J4P2+2 )*( DNM2 / Z( J4-2 ) ) - TAU
          DMIN = MIN( DMIN, DNM1 )
*
          DMIN1 = DMIN
          J4 = J4 + 4
          J4P2 = J4 + 2*PP - 1
          Z( J4-2 ) = DNM1 + Z( J4P2 )
          Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
          DN = Z( J4P2+2 )*( DNM1 / Z( J4-2 ) ) - TAU
          DMIN = MIN( DMIN, DN )
*
        ELSE
*
*          Code for non IEEE arithmetic.
*
          IF( PP.EQ.0 ) THEN
            DO 30 J4 = 4*I0, 4*( NO-3 ), 4
              Z( J4-2 ) = D + Z( J4-1 )
              IF( D.LT.ZERO ) THEN
                RETURN
              ELSE
                Z( J4 ) = Z( J4+1 )*( Z( J4-1 ) / Z( J4-2 ) )
                D = Z( J4+1 )*( D / Z( J4-2 ) ) - TAU
              END IF
              DMIN = MIN( DMIN, D )
              EMIN = MIN( EMIN, Z( J4 ) )
30          CONTINUE
          ELSE
            DO 40 J4 = 4*I0, 4*( NO-3 ), 4
              Z( J4-3 ) = D + Z( J4 )
              IF( D.LT.ZERO ) THEN
                RETURN
              ELSE
                Z( J4-1 ) = Z( J4+2 )*( Z( J4 ) / Z( J4-3 ) )
                D = Z( J4+2 )*( D / Z( J4-3 ) ) - TAU
              END IF
              DMIN = MIN( DMIN, D )
              EMIN = MIN( EMIN, Z( J4-1 ) )

```

```

40      CONTINUE
      END IF
*
*      Unroll last two steps.
*
      DNM2 = D
      DMIN2 = DMIN
      J4 = 4*( NO-2 ) - PP
      J4P2 = J4 + 2*PP - 1
      Z( J4-2 ) = DNM2 + Z( J4P2 )
      IF( DNM2.LT.ZERO ) THEN
          RETURN
      ELSE
          Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
          DNM1 = Z( J4P2+2 )*( DNM2 / Z( J4-2 ) ) - TAU
      END IF
      DMIN = MIN( DMIN, DNM1 )
*
      DMIN1 = DMIN
      J4 = J4 + 4
      J4P2 = J4 + 2*PP - 1
      Z( J4-2 ) = DNM1 + Z( J4P2 )
      IF( DNM1.LT.ZERO ) THEN
          RETURN
      ELSE
          Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
          DN = Z( J4P2+2 )*( DNM1 / Z( J4-2 ) ) - TAU
      END IF
      DMIN = MIN( DMIN, DN )
*
      END IF
*
      Z( J4+2 ) = DN
      Z( 4*NO-PP ) = EMIN
      RETURN
*
*      End of DLASQ5
*
      END

```

— LAPACK dlasq5 —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq5 (i0 n0 z pp tau dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2 ieee)
    (declare (type (member t nil) ieee)

```

```

      (type (double-float) dnm2 dnm1 dn dmin2 f2cl-lib:dmin1 dmin tau)
      (type (simple-array double-float (*)) z)
      (type fixnum pp n0 i0))
(f2cl-lib:with-multi-array-data
  ((z double-float z-%data% z-%offset%))
  (prog ((d 0.0) (emin 0.0) (temp 0.0) (j4 0) (j4p2 0))
    (declare (type (double-float) d emin temp)
      (type fixnum j4 j4p2))
    (if (<= (f2cl-lib:int-sub n0 i0 1) 0) (go end_label))
    (setf j4
      (f2cl-lib:int-sub (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
        3))

    (setf emin
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 4))
        ((1 *))
        z-%offset%))

    (setf d (- (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) tau))
    (setf dmin d)
    (setf f2cl-lib:dmin1
      (- (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))

    (cond
      (ieee
        (cond
          ((= pp 0)
            (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
              (> j4
                (f2cl-lib:int-mul 4
                  (f2cl-lib:int-add n0
                    (f2cl-lib:int-sub
                      3))))))

            nil)

          (tagbody
            (setf (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub j4 2))
              ((1 *))
              z-%offset%)
              (+ d
                (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub j4 1))
                  ((1 *))
                  z-%offset%)))

            (setf temp
              (/
                (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-add j4 1))
                  ((1 *))
                  z-%offset%)
                (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub j4 2))

```

```

((1 *))
z-%offset%)))
(setf d (- (* d temp) tau))
(setf dmin (min dmin d))
(setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
  (*
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub j4 1))
      ((1 *))
      z-%offset%)
    temp))
(setf emin
  (min (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
    emin))))
(t
(f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
  (> j4
    (f2cl-lib:int-mul 4
      (f2cl-lib:int-add n0
        (f2cl-lib:int-sub
          3))))
  nil)
(tagbody
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub j4 3))
    ((1 *))
    z-%offset%)
    (+ d
      (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
  (setf temp
    (/
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 2))
        ((1 *))
        z-%offset%)
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub j4 3))
        ((1 *))
        z-%offset%)))
  (setf d (- (* d temp) tau))
  (setf dmin (min dmin d))
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub j4 1))
    ((1 *))
    z-%offset%)
    (* (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
      temp))
  (setf emin
    (min
      (f2cl-lib:fref z-%data%

```

```

((f2cl-lib:int-sub j4 1))
((1 *))
z-%offset%)
emin))))))
(setf dnm2 d)
(setf dmin2 dmin)
(setf j4
  (f2cl-lib:int-sub
    (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
    pp))
(setf j4p2
  (f2cl-lib:int-sub
    (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
    1))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub j4 2))
  ((1 *))
  z-%offset%)
  (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
  (*
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add j4p2 2))
      ((1 *))
      z-%offset%)
    (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub j4 2))
        ((1 *))
        z-%offset%))))))
(setf dnm1
  (-
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4p2 2))
        ((1 *))
        z-%offset%)
      (/ dnm2
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))
          ((1 *))
          z-%offset%))))
    tau))
(setf dmin (min dmin dnm1))
(setf f2cl-lib:dmin1 dmin)
(setf j4 (f2cl-lib:int-add j4 4))
(setf j4p2
  (f2cl-lib:int-sub
    (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
    1))

```

```

(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub j4 2))
                    ((1 *))
                    z-%offset%))
  (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
  (*
    (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-add j4p2 2))
                  ((1 *))
                  z-%offset%)
    (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
       (f2cl-lib:fref z-%data%
                     ((f2cl-lib:int-sub j4 2))
                     ((1 *))
                     z-%offset%))))))
(setf dn
  (-
    (*
      (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-add j4p2 2))
                    ((1 *))
                    z-%offset%)
      (/ dnm1
         (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-sub j4 2))
                       ((1 *))
                       z-%offset%))))
    tau))
(setf dmin (min dmin dn))
(t
  (cond
    ((= pp 0)
     (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
                   ((> j4
                     (f2cl-lib:int-mul 4
                                         (f2cl-lib:int-add n0
                                                             (f2cl-lib:int-sub
                                                                3))))
                     nil)
     (tagbody
      (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 2))
                        ((1 *))
                        z-%offset%)
        (+ d
          (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 1))
                        ((1 *))
                        z-%offset%))))

```

```

(cond
  (< d zero)
  (go end_label))
(t
  (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 1))
        ((1 *))
        z-%offset%)
      (/
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 1))
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))
          ((1 *))
          z-%offset%))))))
  (setf d
    (-
      (*
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-add j4 1))
          ((1 *))
          z-%offset%)
        (/ d
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub j4 2))
            ((1 *))
            z-%offset%))))))
    tau))))
(setf dmin (min dmin d))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data%
      (j4)
      ((1 *))
      z-%offset%))))))
(t
  (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
    (> j4
      (f2cl-lib:int-mul 4
        (f2cl-lib:int-add n0
          (f2cl-lib:int-sub
            3))))
    nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub j4 3))

```



```

                                ((1 *))
                                z-%offset%)
        (+ d
          (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
(cond
  ((< d zero)
   (go end_label))
  (t
   (setf (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 1))
                        ((1 *))
                        z-%offset%)
          (*
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add j4 2))
                          ((1 *))
                          z-%offset%)
            (/
              (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
              (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub j4 3))
                            ((1 *))
                            z-%offset%))))))
   (setf d
     (-
      (*
        (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-add j4 2))
                      ((1 *))
                      z-%offset%)
        (/ d
          (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-sub j4 3))
                        ((1 *))
                        z-%offset%))))
      tau))))
(setf dmin (min dmin d))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data%
                  ((f2cl-lib:int-sub j4 1))
                  ((1 *))
                  z-%offset%))))))
(setf dnm2 d)
(setf dmin2 dmin)
(setf j4
  (f2cl-lib:int-sub
   (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
   pp))
(setf j4p2

```

```

        (f2cl-lib:int-sub
          (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
          1))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub j4 2))
                    ((1 *))
                    z-%offset%))
      (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
  (< dnm2 zero)
  (go end_label))
(t
  (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
        (*
          (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-add j4p2 2))
                        ((1 *))
                        z-%offset%)
          (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
             (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub j4 2))
                           ((1 *))
                           z-%offset%))))))
  (setf dnm1
        (-
          (*
            (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-add j4p2 2))
                          ((1 *))
                          z-%offset%)
            (/ dnm2
              (f2cl-lib:fref z-%data%
                            ((f2cl-lib:int-sub j4 2))
                            ((1 *))
                            z-%offset%))))
          tau))))
(setf dmin (min dmin dnm1))
(setf f2cl-lib:dmin1 dmin)
(setf j4 (f2cl-lib:int-add j4 4))
(setf j4p2
      (f2cl-lib:int-sub
        (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
        1))
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub j4 2))
                    ((1 *))
                    z-%offset%))
      (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
  (< dnm1 zero)

```

```

      (go end_label))
    (t
      (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
        (*
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-add j4p2 2))
            ((1 *))
            z-%offset%)
          (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-sub j4 2))
              ((1 *))
              z-%offset%))))))
      (setf dn
        (-
          (*
            (f2cl-lib:fref z-%data%
              ((f2cl-lib:int-add j4p2 2))
              ((1 *))
              z-%offset%)
            (/ dnm1
              (f2cl-lib:fref z-%data%
                ((f2cl-lib:int-sub j4 2))
                ((1 *))
                z-%offset%))))
          tau))))
      (setf dmin (min dmin dn))))
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add j4 2))
      ((1 *))
      z-%offset%)
      dn)
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) pp))
      ((1 *))
      z-%offset%)
      emin)
  end_label
  (return
    (values nil
      nil
      nil
      nil
      nil
      dmin
      f2cl-lib:dmin1
      dmin2
      dn
      dnm1
      dnm2

```

```
nil))))))
```

dlasq6 LAPACK

— dlasq6.input —

```
)set break resume
)sys rm -f dlasq6.output
)spool dlasq6.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlasq6.help —

```
=====
dlasq6 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASQ6 - one dqd (shift equal to zero) transform in ping-pong form,
with protection against underflow and overflow

SYNOPSIS

```
SUBROUTINE DLASQ6( IO, NO, Z, PP, DMIN, DMIN1, DMIN2, DN, DNM1, DNM2 )
```

```
INTEGER          IO, NO, PP
```

```
DOUBLE           PRECISION DMIN, DMIN1, DMIN2, DN, DNM1, DNM2
```

```
DOUBLE           PRECISION Z( * )
```

Purpose

```
=====
```

DLASQ6 computes one dqd (shift equal to zero) transform in ping-pong form, with protection against underflow and overflow.

Arguments
=====

IO (input) INTEGER
 First index.

NO (input) INTEGER
 Last index.

Z (input) DOUBLE PRECISION array, dimension (4*N)
 Z holds the qd array. EMIN is stored in Z(4*NO) to avoid
 an extra argument.

PP (input) INTEGER
 PP=0 for ping, PP=1 for pong.

DMIN (output) DOUBLE PRECISION
 Minimum value of d.

DMIN1 (output) DOUBLE PRECISION
 Minimum value of d, excluding D(NO).

DMIN2 (output) DOUBLE PRECISION
 Minimum value of d, excluding D(NO) and D(NO-1).

DN (output) DOUBLE PRECISION
 d(NO), the last value of d.

DNM1 (output) DOUBLE PRECISION
 d(NO-1).

DNM2 (output) DOUBLE PRECISION
 d(NO-2).

— dlasq6.f —

```

      SUBROUTINE DLASQ6( IO, NO, Z, PP, DMIN, DMIN1, DMIN2, DN,
$                      DNM1, DNM2 )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University

```

```

*      October 31, 1999
*
*      .. Scalar Arguments ..
      INTEGER          IO, NO, PP
      DOUBLE PRECISION DMIN, DMIN1, DMIN2, DN, DNM1, DNM2
*
*      ..
*      .. Array Arguments ..
      DOUBLE PRECISION Z( * )
*
*      ..
*
*      =====
*
*      .. Parameter ..
      DOUBLE PRECISION ZERO
      PARAMETER        ( ZERO = 0.0D0 )
*
*      ..
*      .. Local Scalars ..
      INTEGER          J4, J4P2
      DOUBLE PRECISION D, EMIN, SAFMIN, TEMP
*
*      ..
*      .. External Function ..
      DOUBLE PRECISION DLAMCH
      EXTERNAL          DLAMCH
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          MIN
*
*      ..
*      .. Executable Statements ..
*
      IF( ( NO-IO-1 ).LE.0 )
$      RETURN
*
      SAFMIN = DLAMCH( 'Safe minimum' )
      J4 = 4*IO + PP - 3
      EMIN = Z( J4+4 )
      D = Z( J4 )
      DMIN = D
*
      IF( PP.EQ.0 ) THEN
        DO 10 J4 = 4*IO, 4*( NO-3 ), 4
          Z( J4-2 ) = D + Z( J4-1 )
          IF( Z( J4-2 ).EQ.ZERO ) THEN
            Z( J4 ) = ZERO
            D = Z( J4+1 )
            DMIN = D
            EMIN = ZERO
          ELSE IF( SAFMIN*Z( J4+1 ).LT.Z( J4-2 ) .AND.
$           SAFMIN*Z( J4-2 ).LT.Z( J4+1 ) ) THEN
            TEMP = Z( J4+1 ) / Z( J4-2 )
            Z( J4 ) = Z( J4-1 )*TEMP

```

```

        D = D*TEMP
    ELSE
        Z( J4 ) = Z( J4+1 )*( Z( J4-1 ) / Z( J4-2 ) )
        D = Z( J4+1 )*( D / Z( J4-2 ) )
    END IF
    DMIN = MIN( DMIN, D )
    EMIN = MIN( EMIN, Z( J4 ) )
10    CONTINUE
    ELSE
        DO 20 J4 = 4*I0, 4*( NO-3 ), 4
            Z( J4-3 ) = D + Z( J4 )
            IF( Z( J4-3 ).EQ.ZERO ) THEN
                Z( J4-1 ) = ZERO
                D = Z( J4+2 )
                DMIN = D
                EMIN = ZERO
            ELSE IF( SAFMIN*Z( J4+2 ).LT.Z( J4-3 ) .AND.
$           SAFMIN*Z( J4-3 ).LT.Z( J4+2 ) ) THEN
                TEMP = Z( J4+2 ) / Z( J4-3 )
                Z( J4-1 ) = Z( J4 )*TEMP
                D = D*TEMP
            ELSE
                Z( J4-1 ) = Z( J4+2 )*( Z( J4 ) / Z( J4-3 ) )
                D = Z( J4+2 )*( D / Z( J4-3 ) )
            END IF
            DMIN = MIN( DMIN, D )
            EMIN = MIN( EMIN, Z( J4-1 ) )
20    CONTINUE
    END IF
*
*    Unroll last two steps.
*
    DNM2 = D
    DMIN2 = DMIN
    J4 = 4*( NO-2 ) - PP
    J4P2 = J4 + 2*PP - 1
    Z( J4-2 ) = DNM2 + Z( J4P2 )
    IF( Z( J4-2 ).EQ.ZERO ) THEN
        Z( J4 ) = ZERO
        DNM1 = Z( J4P2+2 )
        DMIN = DNM1
        EMIN = ZERO
    ELSE IF( SAFMIN*Z( J4P2+2 ).LT.Z( J4-2 ) .AND.
$   SAFMIN*Z( J4-2 ).LT.Z( J4P2+2 ) ) THEN
        TEMP = Z( J4P2+2 ) / Z( J4-2 )
        Z( J4 ) = Z( J4P2 )*TEMP
        DNM1 = DNM2*TEMP
    ELSE
        Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
        DNM1 = Z( J4P2+2 )*( DNM2 / Z( J4-2 ) )

```

```

      END IF
      DMIN = MIN( DMIN, DNM1 )
*
      DMIN1 = DMIN
      J4 = J4 + 4
      J4P2 = J4 + 2*PP - 1
      Z( J4-2 ) = DNM1 + Z( J4P2 )
      IF( Z( J4-2 ).EQ.ZERO ) THEN
        Z( J4 ) = ZERO
        DN = Z( J4P2+2 )
        DMIN = DN
        EMIN = ZERO
      ELSE IF( SAFMIN*Z( J4P2+2 ).LT.Z( J4-2 ) .AND.
$          SAFMIN*Z( J4-2 ).LT.Z( J4P2+2 ) ) THEN
        TEMP = Z( J4P2+2 ) / Z( J4-2 )
        Z( J4 ) = Z( J4P2 )*TEMP
        DN = DNM1*TEMP
      ELSE
        Z( J4 ) = Z( J4P2+2 )*( Z( J4P2 ) / Z( J4-2 ) )
        DN = Z( J4P2+2 )*( DNM1 / Z( J4-2 ) )
      END IF
      DMIN = MIN( DMIN, DN )
*
      Z( J4+2 ) = DN
      Z( 4*N0-PP ) = EMIN
      RETURN
*
*   End of DLASQ6
*
      END

```

— LAPACK dlasq6 —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlasq6 (i0 n0 z pp dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2)
    (declare (type (double-float) dnm2 dnm1 dn dmin2 f2cl-lib:dmin1 dmin)
              (type (simple-array double-float (*)) z)
              (type fixnum pp n0 i0))
    (f2cl-lib:with-multi-array-data
      ((z double-float z-%data% z-%offset%))
      (prog ((d 0.0) (emin 0.0) (safmin 0.0) (temp 0.0) (j4 0) (j4p2 0))
        (declare (type (double-float) d emin safmin temp)
                  (type fixnum j4 j4p2))
        (if (<= (f2cl-lib:int-sub n0 i0 1) 0) (go end_label))
        (setf safmin (dlamch "Safe minimum"))

```



```

(setf j4
  (f2cl-lib:int-sub (f2cl-lib:int-add (f2cl-lib:int-mul 4 i0) pp)
    3))
(setf emin
  (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-add j4 4))
    ((1 *))
    z-%offset%))
(setf d (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))
(setf dmin d)
(cond
  ((= pp 0)
    (f2cl-lib:fd0 (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
      (> j4
        (f2cl-lib:int-mul 4
          (f2cl-lib:int-add n0
            (f2cl-lib:int-sub
              3))))))
    nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub j4 2))
      ((1 *))
      z-%offset%))
      (+ d
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 1))
          ((1 *))
          z-%offset%)))
    (cond
      ((=
        (f2cl-lib:fref z
          ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
          ((1 *)))
        zero)
        (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
        (setf d
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-add j4 1))
            ((1 *))
            z-%offset%))
          (setf dmin d)
          (setf emin zero))
      (and
        (<
          (* safmin
            (f2cl-lib:fref z ((f2cl-lib:int-add j4 1)) ((1 *))))
          (f2cl-lib:fref z
            ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
            ((1 *))))))

```

```

(<
  (* safmin
    (f2cl-lib:fref z
      ((f2cl-lib:int-add j4
        (f2cl-lib:int-sub 2)))
      ((1 *))))
    (f2cl-lib:fref z ((f2cl-lib:int-add j4 1)) ((1 *))))
  (setf temp
    (/
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 1))
        ((1 *))
        z-%offset%)
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub j4 2))
        ((1 *))
        z-%offset%)))
  (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub j4 1))
        ((1 *))
        z-%offset%)
      temp))
  (setf d (* d temp)))
(t
  (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 1))
        ((1 *))
        z-%offset%)
      (/
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 1))
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))
          ((1 *))
          z-%offset%))))))
  (setf d
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 1))
        ((1 *))
        z-%offset%)
      (/ d
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))

```

```

((1 *))
z-%offset%))))))

(setf dmin (min dmin d))
(setf emin
  (min emin
    (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%))))))
(t
  (f2cl-lib:fdo (j4 (f2cl-lib:int-mul 4 i0) (f2cl-lib:int-add j4 4))
    (> j4
      (f2cl-lib:int-mul 4
        (f2cl-lib:int-add n0
          (f2cl-lib:int-sub
            3))))
    nil)
  (tagbody
    (setf (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub j4 3))
      ((1 *))
      z-%offset%)
      (+ d (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)))
    (cond
      ((=
        (f2cl-lib:fref z
          ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 3)))
          ((1 *)))
        zero)
        (setf (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 1))
          ((1 *))
          z-%offset%)
          zero)
        (setf d
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-add j4 2))
            ((1 *))
            z-%offset%))
          (setf dmin d)
          (setf emin zero))
      ((and
        (<
          (* safmin
            (f2cl-lib:fref z ((f2cl-lib:int-add j4 2)) ((1 *))))
            (f2cl-lib:fref z
              ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 3)))
              ((1 *))))
          (<
            (* safmin
              (f2cl-lib:fref z
                ((f2cl-lib:int-add j4
                  (f2cl-lib:int-sub 3)))

```

```

((1 *)))
(f2cl-lib:fref z ((f2cl-lib:int-add j4 2)) ((1 *))))
(setf temp
  (/
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-add j4 2))
      ((1 *))
      z-%offset%)
    (f2cl-lib:fref z-%data%
      ((f2cl-lib:int-sub j4 3))
      ((1 *))
      z-%offset%)))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub j4 1))
  ((1 *))
  z-%offset%)
  (* (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
    temp))
(setf d (* d temp)))
(t
  (setf (f2cl-lib:fref z-%data%
    ((f2cl-lib:int-sub j4 1))
    ((1 *))
    z-%offset%)
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 2))
        ((1 *))
        z-%offset%)
      (/ (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 3))
          ((1 *))
          z-%offset%))))))
  (setf d
    (*
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4 2))
        ((1 *))
        z-%offset%)
      (/ d
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 3))
          ((1 *))
          z-%offset%))))))
  (setf dmin (min dmin d))
  (setf emin
    (min emin
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-sub j4 1))

```

```

((1 *))
z-%offset%))))))

(setf dnm2 d)
(setf dmin2 dmin)
(setf j4
  (f2cl-lib:int-sub (f2cl-lib:int-mul 4 (f2cl-lib:int-sub n0 2))
    pp))
(setf j4p2
  (f2cl-lib:int-sub (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
    1))
(setf (f2cl-lib:fref z-%data%
  ((f2cl-lib:int-sub j4 2))
  ((1 *))
  z-%offset%)
  (+ dnm2 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
  ((=
    (f2cl-lib:fref z
      ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
      ((1 *)))
    zero)
    (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
    (setf dnm1
      (f2cl-lib:fref z-%data%
        ((f2cl-lib:int-add j4p2 2))
        ((1 *))
        z-%offset%))
    (setf dmin dnm1)
    (setf emin zero))
  (and
    (< (* safmin (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *)))
      (f2cl-lib:fref z
        ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
        ((1 *))))
    (<
      (* safmin
        (f2cl-lib:fref z
          ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
          ((1 *)))
        (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *))))))
    (setf temp
      (/
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-add j4p2 2))
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))
          ((1 *))
          z-%offset%)))

```

```

(setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%) temp))
(setf dnm1 (* dnm2 temp)))
(t
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
       (*
        (f2cl-lib:fref z-%data%
                        ((f2cl-lib:int-add j4p2 2))
                        ((1 *))
                        z-%offset%)
        (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
           (f2cl-lib:fref z-%data%
                           ((f2cl-lib:int-sub j4 2))
                           ((1 *))
                           z-%offset%))))))
(setf dnm1
      (*
       (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add j4p2 2))
                       ((1 *))
                       z-%offset%)
       (/ dnm2
          (f2cl-lib:fref z-%data%
                          ((f2cl-lib:int-sub j4 2))
                          ((1 *))
                          z-%offset%))))))
(setf dmin (min dmin dnm1))
(setf f2cl-lib:dmin1 dmin)
(setf j4 (f2cl-lib:int-add j4 4))
(setf j4p2
      (f2cl-lib:int-sub (f2cl-lib:int-add j4 (f2cl-lib:int-mul 2 pp))
                        1))
(setf (f2cl-lib:fref z-%data%
                      ((f2cl-lib:int-sub j4 2))
                      ((1 *))
                      z-%offset%)
      (+ dnm1 (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)))
(cond
 ((=
  (f2cl-lib:fref z
                  ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
                  ((1 *)))
   zero)
 (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%) zero)
 (setf dn
       (f2cl-lib:fref z-%data%
                       ((f2cl-lib:int-add j4p2 2))
                       ((1 *))
                       z-%offset%)))
 (setf dmin dn)

```

```

      (setf emin zero))
    (and
      (< (* safmin (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *))))
        (f2cl-lib:fref z
          ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
          ((1 *))))
      (<
        (* safmin
          (f2cl-lib:fref z
            ((f2cl-lib:int-add j4 (f2cl-lib:int-sub 2)))
            ((1 *))))
          (f2cl-lib:fref z ((f2cl-lib:int-add j4p2 2)) ((1 *))))))
    (setf temp
      (/
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-add j4p2 2))
          ((1 *))
          z-%offset%)
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-sub j4 2))
          ((1 *))
          z-%offset%)))
    (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
      (* (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%) temp))
    (setf dn (* dnm1 temp)))
  (t
    (setf (f2cl-lib:fref z-%data% (j4) ((1 *)) z-%offset%)
      (*
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-add j4p2 2))
          ((1 *))
          z-%offset%)
        (/ (f2cl-lib:fref z-%data% (j4p2) ((1 *)) z-%offset%)
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub j4 2))
            ((1 *))
            z-%offset%))))))
    (setf dn
      (*
        (f2cl-lib:fref z-%data%
          ((f2cl-lib:int-add j4p2 2))
          ((1 *))
          z-%offset%)
        (/ dnm1
          (f2cl-lib:fref z-%data%
            ((f2cl-lib:int-sub j4 2))
            ((1 *))
            z-%offset%))))))
    (setf dmin (min dmin dn))
    (setf (f2cl-lib:fref z-%data%

```

```

                                ((f2cl-lib:int-add j4 2))
                                ((1 *))
                                z-%offset%)
                                dn)
(setf (f2cl-lib:fref z-%data%
                    ((f2cl-lib:int-sub (f2cl-lib:int-mul 4 n0) pp))
                    ((1 *))
                    z-%offset%)
      emin)
end_label
(return
 (values nil nil nil nil dmin f2cl-lib:dmin1 dmin2 dn dnm1 dnm2))))))

```

dlasr LAPACK

— dlasr.input —

```

)set break resume
)sys rm -f dlasr.output
)spool dlasr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasr.help —

```

=====
dlasr examples
=====

=====
Man Page Details
=====

```

NAME

DLASR - a sequence of plane rotations to a real matrix A,

SYNOPSIS


```
SUBROUTINE DLASR( SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA )
```

```
    CHARACTER    DIRECT, PIVOT, SIDE
```

```
    INTEGER      LDA, M, N
```

```
    DOUBLE       PRECISION A( LDA, * ), C( * ), S( * )
```

Purpose

=====

DLASR performs the transformation

$A := P \cdot A$, when SIDE = 'L' or 'l' (Left-hand side)

$A := A \cdot P$, when SIDE = 'R' or 'r' (Right-hand side)

where A is an m by n real matrix and P is an orthogonal matrix, consisting of a sequence of plane rotations determined by the parameters PIVOT and DIRECT as follows (z = m when SIDE = 'L' or 'l' and z = n when SIDE = 'R' or 'r'):

When DIRECT = 'F' or 'f' (Forward sequence) then

$P = P(z - 1) * \dots * P(2) * P(1),$

and when DIRECT = 'B' or 'b' (Backward sequence) then

$P = P(1) * P(2) * \dots * P(z - 1),$

where $P(k)$ is a plane rotation matrix for the following planes:

when PIVOT = 'V' or 'v' (Variable pivot),
the plane (k, k + 1)

when PIVOT = 'T' or 't' (Top pivot),
the plane (1, k + 1)

when PIVOT = 'B' or 'b' (Bottom pivot),
the plane (k, z)

$c(k)$ and $s(k)$ must contain the cosine and sine that define the matrix $P(k)$. The two by two plane rotation part of the matrix $P(k)$, $R(k)$, is assumed to be of the form

$$R(k) = \begin{pmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{pmatrix}.$$

This version vectorises across rows of the array A when SIDE = 'L'.

Arguments

=====

SIDE (input) CHARACTER*1
 Specifies whether the plane rotation matrix P is applied to A on the left or the right.
 = 'L': Left, compute A := P*A
 = 'R': Right, compute A:= A*P'

DIRECT (input) CHARACTER*1
 Specifies whether P is a forward or backward sequence of plane rotations.
 = 'F': Forward, $P = P(z-1) \dots P(2) P(1)$
 = 'B': Backward, $P = P(1) P(2) \dots P(z-1)$

PIVOT (input) CHARACTER*1
 Specifies the plane for which P(k) is a plane rotation matrix.
 = 'V': Variable pivot, the plane (k,k+1)
 = 'T': Top pivot, the plane (1,k+1)
 = 'B': Bottom pivot, the plane (k,z)

M (input) INTEGER
 The number of rows of the matrix A. If $m \leq 1$, an immediate return is effected.

N (input) INTEGER
 The number of columns of the matrix A. If $n \leq 1$, an immediate return is effected.

C, S (input) DOUBLE PRECISION arrays, dimension
 (M-1) if SIDE = 'L'
 (N-1) if SIDE = 'R'
 c(k) and s(k) contain the cosine and sine that define the matrix P(k). The two by two plane rotation part of the matrix P(k), R(k), is assumed to be of the form

$$R(k) = \begin{pmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{pmatrix}.$$

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 The m by n matrix A. On exit, A is overwritten by P*A if SIDE = 'R' or by A*P' if SIDE = 'L'.

LDA (input) INTEGER
 The leading dimension of the array A. $LDA \geq \max(1,M)$.

— dlasr.f —

```

SUBROUTINE DLASR( SIDE, PIVOT, DIRECT, M, N, C, S, A, LDA )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  October 31, 1992
*
*  .. Scalar Arguments ..
CHARACTER      DIRECT, PIVOT, SIDE
INTEGER        LDA, M, N
*
*  ..
*  .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), C( * ), S( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION  ONE, ZERO
PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*  ..
*  .. Local Scalars ..
INTEGER          I, INFO, J
DOUBLE PRECISION  CTEMP, STEMP, TEMP
*
*  ..
*  .. External Functions ..
LOGICAL          LSAME
EXTERNAL         LSAME
*
*  ..
*  .. External Subroutines ..
EXTERNAL         XERBLA
*
*  ..
*  .. Intrinsic Functions ..
INTRINSIC        MAX
*
*  ..
*  .. Executable Statements ..
*
*  Test the input parameters
*
*
*  INFO = 0
*  IF( .NOT.( LSAME( SIDE, 'L' ) .OR. LSAME( SIDE, 'R' ) ) ) THEN
*      INFO = 1
*  ELSE IF( .NOT.( LSAME( PIVOT, 'V' ) .OR. LSAME( PIVOT,
$      'T' ) .OR. LSAME( PIVOT, 'B' ) ) ) THEN
*      INFO = 2
*  ELSE IF( .NOT.( LSAME( DIRECT, 'F' ) .OR. LSAME( DIRECT, 'B' ) ) )
$      THEN
*      INFO = 3

```

```

ELSE IF( M.LT.0 ) THEN
    INFO = 4
ELSE IF( N.LT.0 ) THEN
    INFO = 5
ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
    INFO = 9
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DLASR ', INFO )
    RETURN
END IF

*
* Quick return if possible
*
IF( ( M.EQ.0 ) .OR. ( N.EQ.0 ) )
$ RETURN
IF( LSAME( SIDE, 'L' ) ) THEN

*
* Form P * A
*
IF( LSAME( PIVOT, 'V' ) ) THEN
    IF( LSAME( DIRECT, 'F' ) ) THEN
        DO 20 J = 1, M - 1
            CTEMP = C( J )
            STEMP = S( J )
            IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                DO 10 I = 1, N
                    TEMP = A( J+1, I )
                    A( J+1, I ) = CTEMP*TEMP - STEMP*A( J, I )
                    A( J, I ) = STEMP*TEMP + CTEMP*A( J, I )
20                CONTINUE
            END IF
        CONTINUE
    ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
        DO 40 J = M - 1, 1, -1
            CTEMP = C( J )
            STEMP = S( J )
            IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                DO 30 I = 1, N
                    TEMP = A( J+1, I )
                    A( J+1, I ) = CTEMP*TEMP - STEMP*A( J, I )
                    A( J, I ) = STEMP*TEMP + CTEMP*A( J, I )
30                CONTINUE
            END IF
        CONTINUE
    ELSE IF( LSAME( PIVOT, 'T' ) ) THEN
        IF( LSAME( DIRECT, 'F' ) ) THEN
            DO 60 J = 2, M
                CTEMP = C( J-1 )

```

```

        STEMP = S( J-1 )
        IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
            DO 50 I = 1, N
                TEMP = A( J, I )
                A( J, I ) = CTEMP*TEMP - STEMP*A( 1, I )
                A( 1, I ) = STEMP*TEMP + CTEMP*A( 1, I )
50          CONTINUE
            END IF
60          CONTINUE
        ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
            DO 80 J = M, 2, -1
                CTEMP = C( J-1 )
                STEMP = S( J-1 )
                IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                    DO 70 I = 1, N
                        TEMP = A( J, I )
                        A( J, I ) = CTEMP*TEMP - STEMP*A( 1, I )
                        A( 1, I ) = STEMP*TEMP + CTEMP*A( 1, I )
70          CONTINUE
                    END IF
80          CONTINUE
            END IF
        ELSE IF( LSAME( PIVOT, 'B' ) ) THEN
            IF( LSAME( DIRECT, 'F' ) ) THEN
                DO 100 J = 1, M - 1
                    CTEMP = C( J )
                    STEMP = S( J )
                    IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                        DO 90 I = 1, N
                            TEMP = A( J, I )
                            A( J, I ) = STEMP*A( M, I ) + CTEMP*TEMP
                            A( M, I ) = CTEMP*A( M, I ) - STEMP*TEMP
90          CONTINUE
                        END IF
100         CONTINUE
            ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
                DO 120 J = M - 1, 1, -1
                    CTEMP = C( J )
                    STEMP = S( J )
                    IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                        DO 110 I = 1, N
                            TEMP = A( J, I )
                            A( J, I ) = STEMP*A( M, I ) + CTEMP*TEMP
                            A( M, I ) = CTEMP*A( M, I ) - STEMP*TEMP
110         CONTINUE
                        END IF
120         CONTINUE
            END IF
        ELSE IF( LSAME( SIDE, 'R' ) ) THEN

```

```

*
*      Form A * P'
*
      IF( LSAME( PIVOT, 'V' ) ) THEN
        IF( LSAME( DIRECT, 'F' ) ) THEN
          DO 140 J = 1, N - 1
            CTEMP = C( J )
            STEMP = S( J )
            IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
              DO 130 I = 1, M
                TEMP = A( I, J+1 )
                A( I, J+1 ) = CTEMP*TEMP - STEMP*A( I, J )
                A( I, J ) = STEMP*TEMP + CTEMP*A( I, J )
130              CONTINUE
            END IF
          CONTINUE
140        ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
          DO 160 J = N - 1, 1, -1
            CTEMP = C( J )
            STEMP = S( J )
            IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
              DO 150 I = 1, M
                TEMP = A( I, J+1 )
                A( I, J+1 ) = CTEMP*TEMP - STEMP*A( I, J )
                A( I, J ) = STEMP*TEMP + CTEMP*A( I, J )
150              CONTINUE
            END IF
          CONTINUE
160        ELSE IF
          ELSE IF( LSAME( PIVOT, 'T' ) ) THEN
            IF( LSAME( DIRECT, 'F' ) ) THEN
              DO 180 J = 2, N
                CTEMP = C( J-1 )
                STEMP = S( J-1 )
                IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                  DO 170 I = 1, M
                    TEMP = A( I, J )
                    A( I, J ) = CTEMP*TEMP - STEMP*A( I, 1 )
                    A( I, 1 ) = STEMP*TEMP + CTEMP*A( I, 1 )
170                  CONTINUE
                END IF
              CONTINUE
180            ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
              DO 200 J = N, 2, -1
                CTEMP = C( J-1 )
                STEMP = S( J-1 )
                IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                  DO 190 I = 1, M
                    TEMP = A( I, J )
                    A( I, J ) = CTEMP*TEMP - STEMP*A( I, 1 )

```

```

          A( I, 1 ) = STEMP*TEMP + CTEMP*A( I, 1 )
190      CONTINUE
          END IF
200      CONTINUE
          END IF
          ELSE IF( LSAME( PIVOT, 'B' ) ) THEN
            IF( LSAME( DIRECT, 'F' ) ) THEN
              DO 220 J = 1, N - 1
                CTEMP = C( J )
                STEMP = S( J )
                IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                  DO 210 I = 1, M
                    TEMP = A( I, J )
                    A( I, J ) = STEMP*A( I, N ) + CTEMP*TEMP
                    A( I, N ) = CTEMP*A( I, N ) - STEMP*TEMP
210              CONTINUE
                  END IF
220              CONTINUE
            ELSE IF( LSAME( DIRECT, 'B' ) ) THEN
              DO 240 J = N - 1, 1, -1
                CTEMP = C( J )
                STEMP = S( J )
                IF( ( CTEMP.NE.ONE ) .OR. ( STEMP.NE.ZERO ) ) THEN
                  DO 230 I = 1, M
                    TEMP = A( I, J )
                    A( I, J ) = STEMP*A( I, N ) + CTEMP*TEMP
                    A( I, N ) = CTEMP*A( I, N ) - STEMP*TEMP
230              CONTINUE
                  END IF
240              CONTINUE
            END IF
          END IF
          END IF
          END IF
*
          RETURN
*
*   End of DLASR
*
          END

```

— LAPACK dlasr —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dlasr (side pivot direct m n c s a lda)

```

```

(declare (type (simple-array double-float (*)) a s c)
  (type fixnum lda n m)
  (type character direct pivot side))
(f2cl-lib:with-multi-array-data
  ((side character side-%data% side-%offset%)
   (pivot character pivot-%data% pivot-%offset%)
   (direct character direct-%data% direct-%offset%)
   (c double-float c-%data% c-%offset%)
   (s double-float s-%data% s-%offset%)
   (a double-float a-%data% a-%offset%))
  (prog ((ctemp 0.0) (stemp 0.0) (temp 0.0) (i 0) (info 0) (j 0))
    (declare (type (double-float) ctemp stemp temp)
      (type fixnum i info j))
    (setf info 0)
    (cond
      ((not (or (char-equal side #\L) (char-equal side #\R)))
       (setf info 1))
      ((not (or (char-equal pivot #\V) (char-equal pivot #\T)
                (char-equal pivot #\B)))
       (setf info 2))
      ((not (or (char-equal direct #\F) (char-equal direct #\B)))
       (setf info 3))
      ((< m 0)
       (setf info 4))
      ((< n 0)
       (setf info 5))
      ((< lda (max (the fixnum 1) (the fixnum m)))
       (setf info 9)))
    (cond
      ((/= info 0)
       (error
        " ** On entry to ~a parameter number ~a had an illegal value~%"
        "DLASR" info)
       (go end_label)))
    (if (or (= m 0) (= n 0)) (go end_label))
    (cond
      ((char-equal side #\L)
       (cond
         ((char-equal pivot #\V)
          (cond
            ((char-equal direct #\F)
             (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                           ((> j (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
                            nil)
              (tagbody
                (setf ctemp
                  (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
                (setf stemp
                  (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
                (cond

```



```

((or (/= ctemp one) (/= stemp zero))
 (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  (> i n) nil)
 (tagbody
  (setf temp
    (f2cl-lib:fref a-%data%
      ((f2cl-lib:int-add j 1) i)
      ((1 lda) (1 *))
      a-%offset%))
  (setf (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-add j 1) i)
    ((1 lda) (1 *))
    a-%offset%))
    (- (* ctemp temp)
      (* stemp
        (f2cl-lib:fref a-%data%
          (j i)
          ((1 lda) (1 *))
          a-%offset%))))
  (setf (f2cl-lib:fref a-%data%
    (j i)
    ((1 lda) (1 *))
    a-%offset%))
    (+ (* stemp temp)
      (* ctemp
        (f2cl-lib:fref a-%data%
          (j i)
          ((1 lda) (1 *))
          a-%offset%)))))))))
(char-equal direct #\B)
(f2cl-lib:fdo (j (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
  (> j 1) nil)
 (tagbody
  (setf ctemp
    (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
  (setf stemp
    (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
  (cond
   ((or (/= ctemp one) (/= stemp zero))
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     (> i n) nil)
    (tagbody
     (setf temp
      (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add j 1) i)
        ((1 lda) (1 *))
        a-%offset%))
      (setf (f2cl-lib:fref a-%data%
        ((f2cl-lib:int-add j 1) i)

```

```

((1 lda) (1 *))
a-%offset%)
(- (* ctemp temp)
  (* stemp
    (f2cl-lib:fref a-%data%
      (j i)
      ((1 lda) (1 *))
      a-%offset%))))
(setf (f2cl-lib:fref a-%data%
  (j i)
  ((1 lda) (1 *))
  a-%offset%)
  (+ (* stemp temp)
    (* ctemp
      (f2cl-lib:fref a-%data%
        (j i)
        ((1 lda) (1 *))
        a-%offset%)))))))))

((char-equal pivot #\T)
  (cond
    ((char-equal direct #\F)
      (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
        ((> j m) nil)
        (tagbody
          (setf ctemp
            (f2cl-lib:fref c-%data%
              ((f2cl-lib:int-sub j 1))
              ((1 *))
              c-%offset%))
            (setf stemp
              (f2cl-lib:fref s-%data%
                ((f2cl-lib:int-sub j 1))
                ((1 *))
                s-%offset%))
            (cond
              ((or (/= ctemp one) (/= stemp zero))
                (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
                  (tagbody
                    (setf temp
                      (f2cl-lib:fref a-%data%
                        (j i)
                        ((1 lda) (1 *))
                        a-%offset%))
                    (setf (f2cl-lib:fref a-%data%
                      (j i)
                      ((1 lda) (1 *))
                      a-%offset%)
                      (- (* ctemp temp)
                        (* stemp

```

```

(f2cl-lib:fref a-%data%
  (1 i)
  ((1 lda) (1 *))
  a-%offset%)))
(setf (f2cl-lib:fref a-%data%
  (1 i)
  ((1 lda) (1 *))
  a-%offset%)
  (+ (* stemp temp)
    (* ctemp
      (f2cl-lib:fref a-%data%
        (1 i)
        ((1 lda) (1 *))
        a-%offset%)))))))))
(char-equal direct #\B)
(f2cl-lib:fdo (j m (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j 2) nil)
(tagbody
  (setf ctemp
    (f2cl-lib:fref c-%data%
      ((f2cl-lib:int-sub j 1))
      ((1 *))
      c-%offset%))
    (setf stemp
      (f2cl-lib:fref s-%data%
        ((f2cl-lib:int-sub j 1))
        ((1 *))
        s-%offset%))
    (cond
      ((or (/= ctemp one) (/= stemp zero))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i n) nil)
        (tagbody
          (setf temp
            (f2cl-lib:fref a-%data%
              (j i)
              ((1 lda) (1 *))
              a-%offset%))
            (setf (f2cl-lib:fref a-%data%
              (j i)
              ((1 lda) (1 *))
              a-%offset%)
              (- (* ctemp temp)
                (* stemp
                  (f2cl-lib:fref a-%data%
                    (1 i)
                    ((1 lda) (1 *))
                    a-%offset%))))
            (setf (f2cl-lib:fref a-%data%
              (1 i)
              a-%offset%))))))

```

```

((1 lda) (1 *))
a-%offset%)
(+ (* stemp temp)
  (* ctemp
    (f2cl-lib:fref a-%data%
      (1 i)
      ((1 lda) (1 *))
      a-%offset%)))))))))
((char-equal pivot #\B)
  (cond
    ((char-equal direct #\F)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
          nil)
      (tagbody
        (setf ctemp
          (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
        (setf stemp
          (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
        (cond
          ((or (/= ctemp one) (/= stemp zero))
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i n) nil)
            (tagbody
              (setf temp
                (f2cl-lib:fref a-%data%
                  (j i)
                  ((1 lda) (1 *))
                  a-%offset%))
              (setf (f2cl-lib:fref a-%data%
                (j i)
                ((1 lda) (1 *))
                a-%offset%)
                (+
                  (* stemp
                    (f2cl-lib:fref a-%data%
                      (m i)
                      ((1 lda) (1 *))
                      a-%offset%))
                  (* ctemp temp)))
              (setf (f2cl-lib:fref a-%data%
                (m i)
                ((1 lda) (1 *))
                a-%offset%)
                (-
                  (* ctemp
                    (f2cl-lib:fref a-%data%
                      (m i)
                      ((1 lda) (1 *))
                      a-%offset%))

```

```

(* stemp temp))))))))))
((char-equal direct #\B)
(f2cl-lib:fdo (j (f2cl-lib:int-add m (f2cl-lib:int-sub 1))
(f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
(> j 1) nil)
(tagbody
(setf ctemp
(f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
(setf stemp
(f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
(cond
((or (/= ctemp one) (/= stemp zero))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i n) nil)
(tagbody
(setf temp
(f2cl-lib:fref a-%data%
(j i)
((1 lda) (1 *))
a-%offset%))
(setf (f2cl-lib:fref a-%data%
(j i)
((1 lda) (1 *))
a-%offset%))
(+
(* stemp
(f2cl-lib:fref a-%data%
(m i)
((1 lda) (1 *))
a-%offset%))
(* ctemp temp)))
(setf (f2cl-lib:fref a-%data%
(m i)
((1 lda) (1 *))
a-%offset%))
(-
(* ctemp
(f2cl-lib:fref a-%data%
(m i)
((1 lda) (1 *))
a-%offset%))
(* stemp temp))))))))))))))
((char-equal side #\R)
(cond
(char-equal pivot #\V)
(cond
(char-equal direct #\F)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
nil)

```

```
(tagbody
  (setf ctemp
    (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
  (setf stemp
    (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
  (cond
    ((or (/= ctemp one) (/= stemp zero))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i m) nil)
        (tagbody
          (setf temp
            (f2cl-lib:fref a-%data%
              (i (f2cl-lib:int-add j 1))
              ((1 lda) (1 *))
              a-%offset%))
            (setf (f2cl-lib:fref a-%data%
              (i (f2cl-lib:int-add j 1))
              ((1 lda) (1 *))
              a-%offset%)
              (- (* ctemp temp)
                (* stemp
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%))))))
            (setf (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)
              (+ (* stemp temp)
                (* ctemp
                  (f2cl-lib:fref a-%data%
                    (i j)
                    ((1 lda) (1 *))
                    a-%offset%)))))))))))
((char-equal direct #\B)
  (f2cl-lib:fdo (j (f2cl-lib:int-add n (f2cl-lib:int-sub 1))
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
    ((> j 1) nil)
    (tagbody
      (setf ctemp
        (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
      (setf stemp
        (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
      (cond
        ((or (/= ctemp one) (/= stemp zero))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf temp
```

```

        (f2cl-lib:fref a-%data%
          (i (f2cl-lib:int-add j 1))
          ((1 lda) (1 *))
          a-%offset%))
      (setf (f2cl-lib:fref a-%data%
        (i (f2cl-lib:int-add j 1))
        ((1 lda) (1 *))
        a-%offset%))
        (- (* ctemp temp)
          (* stemp
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%))))))
      (setf (f2cl-lib:fref a-%data%
        (i j)
        ((1 lda) (1 *))
        a-%offset%))
        (+ (* stemp temp)
          (* ctemp
            (f2cl-lib:fref a-%data%
              (i j)
              ((1 lda) (1 *))
              a-%offset%)))))))))
((char-equal pivot #\T)
 (cond
  ((char-equal direct #\F)
   (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
     (> j n) nil)
   (tagbody
    (setf ctemp
      (f2cl-lib:fref c-%data%
        ((f2cl-lib:int-sub j 1))
        ((1 *))
        c-%offset%))
      (setf stemp
        (f2cl-lib:fref s-%data%
          ((f2cl-lib:int-sub j 1))
          ((1 *))
          s-%offset%))
        (cond
         ((or (/= ctemp one) (/= stemp zero))
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            (> i m) nil)
          (tagbody
           (setf temp
             (f2cl-lib:fref a-%data%
               (i j)
               ((1 lda) (1 *))
               a-%offset%))

```

```

      (setf (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%))
      (- (* ctemp temp)
         (* stemp
            (f2cl-lib:fref a-%data%
                          (i 1)
                          ((1 lda) (1 *))
                          a-%offset%))))
      (setf (f2cl-lib:fref a-%data%
                          (i 1)
                          ((1 lda) (1 *))
                          a-%offset%))
      (+ (* stemp temp)
         (* ctemp
            (f2cl-lib:fref a-%data%
                          (i 1)
                          ((1 lda) (1 *))
                          a-%offset%)))))))))
(char-equal direct #\B)
(f2cl-lib:fdo (j n (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j 2) nil)
(tagbody
  (setf ctemp
        (f2cl-lib:fref c-%data%
                        ((f2cl-lib:int-sub j 1))
                        ((1 *))
                        c-%offset%))
    (setf stemp
          (f2cl-lib:fref s-%data%
                        ((f2cl-lib:int-sub j 1))
                        ((1 *))
                        s-%offset%))
    (cond
      ((or (/= ctemp one) (/= stemp zero))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
        (tagbody
          (setf temp
                (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *))
                              a-%offset%))
            (setf (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *))
                              a-%offset%))
              (- (* ctemp temp)
                 (* stemp

```



```

(f2cl-lib:fref a-%data%
  (i 1)
  ((1 lda) (1 *))
  a-%offset%)))
(setf (f2cl-lib:fref a-%data%
  (i 1)
  ((1 lda) (1 *))
  a-%offset%)
  (+ (* stemp temp)
    (* ctemp
      (f2cl-lib:fref a-%data%
        (i 1)
        ((1 lda) (1 *))
        a-%offset%)))))))))
((char-equal pivot #\B)
 (cond
  ((char-equal direct #\F)
   (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
     ((> j (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (setf ctemp
        (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
      (setf stemp
        (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
      (cond
        ((or (/= ctemp one) (/= stemp zero))
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
           ((> i m) nil)
          (tagbody
            (setf temp
              (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%))
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
                (+
                  (* stemp
                    (f2cl-lib:fref a-%data%
                      (i n)
                      ((1 lda) (1 *))
                      a-%offset%))
                  (* ctemp temp)))
              (setf (f2cl-lib:fref a-%data%
                (i n)
                ((1 lda) (1 *))
                a-%offset%))

```

```

(-
  (* ctemp
    (f2cl-lib:fref a-%data%
                   (i n)
                   ((1 lda) (1 *))
                   a-%offset%))
    (* stemp temp)))))))))
(char-equal direct #\B)
(f2cl-lib:fdo (j (f2cl-lib:int-add n (f2cl-lib:int-sub 1))
              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              (> j 1) nil)
(tagbody
  (setf ctemp
        (f2cl-lib:fref c-%data% (j) ((1 *)) c-%offset%))
  (setf stemp
        (f2cl-lib:fref s-%data% (j) ((1 *)) s-%offset%))
  (cond
    ((or (/= ctemp one) (/= stemp zero))
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   (> i m) nil)
     (tagbody
       (setf temp
              (f2cl-lib:fref a-%data%
                             (i j)
                             ((1 lda) (1 *))
                             a-%offset%))
       (setf (f2cl-lib:fref a-%data%
                           (i j)
                           ((1 lda) (1 *))
                           a-%offset%))
              (+
                (* stemp
                  (f2cl-lib:fref a-%data%
                                 (i n)
                                 ((1 lda) (1 *))
                                 a-%offset%))
                (* ctemp temp)))
       (setf (f2cl-lib:fref a-%data%
                           (i n)
                           ((1 lda) (1 *))
                           a-%offset%))
              (-
                (* ctemp
                  (f2cl-lib:fref a-%data%
                                 (i n)
                                 ((1 lda) (1 *))
                                 a-%offset%))
                (* stemp temp)))))))))
end_label
(return (values nil nil nil nil nil nil nil nil))))))

```

dlasrt LAPACK

— dlasrt.input —

```
)set break resume
)sys rm -f dlasrt.output
)spool dlasrt.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlasrt.help —

```
=====
dlasrt examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DLASRT - number in D in increasing order (if ID = 'I') or in decreasing order (if ID = 'D')

SYNOPSIS

```
SUBROUTINE DLASRT( ID, N, D, INFO )
```

| | |
|-----------|------------------|
| CHARACTER | ID |
| INTEGER | INFO, N |
| DOUBLE | PRECISION D(*) |

Purpose

```
=====
```

Sort the numbers in D in increasing order (if ID = 'I') or in decreasing order (if ID = 'D').

Use Quick Sort, reverting to Insertion sort on arrays of size ≤ 20 . Dimension of STACK limits N to about $2^{**}32$.

Arguments

=====

ID (input) CHARACTER*1
 = 'I': sort D in increasing order;
 = 'D': sort D in decreasing order.

N (input) INTEGER
 The length of the array D.

D (input/output) DOUBLE PRECISION array, dimension (N)
 On entry, the array to be sorted.
 On exit, D has been sorted into increasing order
 ($D(1) \leq \dots \leq D(N)$) or into decreasing order
 ($D(1) \geq \dots \geq D(N)$), depending on ID.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

— dlasrt.f —

```

      SUBROUTINE DLASRT( ID, N, D, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  September 30, 1994
*
*  .. Scalar Arguments ..
*     CHARACTER          ID
*     INTEGER            INFO, N
*  ..
*  .. Array Arguments ..
*     DOUBLE PRECISION   D( * )
*  ..
*
*  =====
*
*  .. Parameters ..

```

```

      INTEGER          SELECT
      PARAMETER        ( SELECT = 20 )
*
*   ..
*   .. Local Scalars ..
      INTEGER          DIR, ENDD, I, J, START, STKPNT
      DOUBLE PRECISION D1, D2, D3, DMNMX, TMP
*
*   ..
*   .. Local Arrays ..
      INTEGER          STACK( 2, 32 )
*
*   ..
*   .. External Functions ..
      LOGICAL          LSAME
      EXTERNAL          LSAME
*
*   ..
*   .. External Subroutines ..
      EXTERNAL          XERBLA
*
*   ..
*   .. Executable Statements ..
*
*   Test the input paramters.
*
      INFO = 0
      DIR = -1
      IF( LSAME( ID, 'D' ) ) THEN
         DIR = 0
      ELSE IF( LSAME( ID, 'I' ) ) THEN
         DIR = 1
      END IF
      IF( DIR.EQ.-1 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DLASRT', -INFO )
         RETURN
      END IF
*
*   Quick return if possible
*
      IF( N.LE.1 )
$    RETURN
*
      STKPNT = 1
      STACK( 1, 1 ) = 1
      STACK( 2, 1 ) = N
10  CONTINUE
      START = STACK( 1, STKPNT )
      ENDD = STACK( 2, STKPNT )
      STKPNT = STKPNT - 1

```

```

      IF( ENDD-START.LE.SELECT .AND. ENDD-START.GT.0 ) THEN
*
*      Do Insertion sort on D( START:ENDD )
*
      IF( DIR.EQ.0 ) THEN
*
*      Sort into decreasing order
*
        DO 30 I = START + 1, ENDD
          DO 20 J = I, START + 1, -1
            IF( D( J ).GT.D( J-1 ) ) THEN
              DMNMX = D( J )
              D( J ) = D( J-1 )
              D( J-1 ) = DMNMX
            ELSE
              GO TO 30
            END IF
          CONTINUE
        30 CONTINUE
*
      ELSE
*
*      Sort into increasing order
*
        DO 50 I = START + 1, ENDD
          DO 40 J = I, START + 1, -1
            IF( D( J ).LT.D( J-1 ) ) THEN
              DMNMX = D( J )
              D( J ) = D( J-1 )
              D( J-1 ) = DMNMX
            ELSE
              GO TO 50
            END IF
          CONTINUE
        40 CONTINUE
        50 CONTINUE
*
      END IF
*
      ELSE IF( ENDD-START.GT.SELECT ) THEN
*
*      Partition D( START:ENDD ) and stack parts, largest one first
*
*      Choose partition entry as median of 3
*
        D1 = D( START )
        D2 = D( ENDD )
        I = ( START+ENDD ) / 2
        D3 = D( I )
        IF( D1.LT.D2 ) THEN
          IF( D3.LT.D1 ) THEN

```

```

        DMNMX = D1
      ELSE IF( D3.LT.D2 ) THEN
        DMNMX = D3
      ELSE
        DMNMX = D2
      END IF
    ELSE
      IF( D3.LT.D2 ) THEN
        DMNMX = D2
      ELSE IF( D3.LT.D1 ) THEN
        DMNMX = D3
      ELSE
        DMNMX = D1
      END IF
    END IF
  *
  IF( DIR.EQ.0 ) THEN
  *
  *      Sort into decreasing order
  *
    I = START - 1
    J = ENDD + 1
60    CONTINUE
70    CONTINUE
    J = J - 1
    IF( D( J ).LT.DMNMX )
      $      GO TO 70
80    CONTINUE
    I = I + 1
    IF( D( I ).GT.DMNMX )
      $      GO TO 80
    IF( I.LT.J ) THEN
      TMP = D( I )
      D( I ) = D( J )
      D( J ) = TMP
      GO TO 60
    END IF
    IF( J-START.GT.ENDD-J-1 ) THEN
      STKPNT = STKPNT + 1
      STACK( 1, STKPNT ) = START
      STACK( 2, STKPNT ) = J
      STKPNT = STKPNT + 1
      STACK( 1, STKPNT ) = J + 1
      STACK( 2, STKPNT ) = ENDD
    ELSE
      STKPNT = STKPNT + 1
      STACK( 1, STKPNT ) = J + 1
      STACK( 2, STKPNT ) = ENDD
      STKPNT = STKPNT + 1
      STACK( 1, STKPNT ) = START
    
```

```

        STACK( 2, STKPNT ) = J
    END IF
ELSE
*
*      Sort into increasing order
*
        I = START - 1
        J = ENDD + 1
90      CONTINUE
100     CONTINUE
        J = J - 1
        IF( D( J ).GT.DMNMX )
            $      GO TO 100
110     CONTINUE
        I = I + 1
        IF( D( I ).LT.DMNMX )
            $      GO TO 110
        IF( I.LT.J ) THEN
            TMP = D( I )
            D( I ) = D( J )
            D( J ) = TMP
            GO TO 90
        END IF
        IF( J-START.GT.ENDD-J-1 ) THEN
            STKPNT = STKPNT + 1
            STACK( 1, STKPNT ) = START
            STACK( 2, STKPNT ) = J
            STKPNT = STKPNT + 1
            STACK( 1, STKPNT ) = J + 1
            STACK( 2, STKPNT ) = ENDD
        ELSE
            STKPNT = STKPNT + 1
            STACK( 1, STKPNT ) = J + 1
            STACK( 2, STKPNT ) = ENDD
            STKPNT = STKPNT + 1
            STACK( 1, STKPNT ) = START
            STACK( 2, STKPNT ) = J
        END IF
    END IF
    IF( STKPNT.GT.0 )
        $      GO TO 10
    RETURN
*
*      End of DLASRT
*
END

```

— LAPACK dlasrt —

```

(let* ((select 20))
  (declare (type (fixnum 20 20) select))
  (defun dlasrt (id n d info)
    (declare (type (simple-array double-float (*)) d)
              (type fixnum info n)
              (type character id))
    (f2cl-lib:with-multi-array-data
      ((id character id-%data% id-%offset%)
       (d double-float d-%data% d-%offset%))
      (prog ((stack (make-array 64 :element-type 'fixnum)) (d1 0.0)
              (d2 0.0) (d3 0.0) (dmnmx 0.0) (tmp 0.0) (dir 0) (endd 0) (i 0)
              (j 0) (start 0) (stkpnt 0))
        (declare (type (simple-array fixnum (64)) stack)
                  (type (double-float) d1 d2 d3 dmnmx tmp)
                  (type fixnum dir endd i j start stkpnt))
        (setf info 0)
        (setf dir -1)
        (cond
          ((char-equal id #\D)
           (setf dir 0))
          ((char-equal id #\I)
           (setf dir 1)))
        (cond
          ((= dir (f2cl-lib:int-sub 1))
           (setf info -1))
          ((< n 0)
           (setf info -2)))
        (cond
          ((/= info 0)
           (error
            " ** On entry to ~a parameter number ~a had an illegal value~%"
            "DLASRT" (f2cl-lib:int-sub info))
           (go end_label)))
        (if (<= n 1) (go end_label))
        (setf stkpnt 1)
        (setf (f2cl-lib:fref stack (1 1) ((1 2) (1 32))) 1)
        (setf (f2cl-lib:fref stack (2 1) ((1 2) (1 32))) n)
label10
        (setf start (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))))
        (setf endd (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))))
        (setf stkpnt (f2cl-lib:int-sub stkpnt 1))
        (cond
          ((and (<= (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) select)
                (> (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) 0))
           (cond
            ((= dir 0)
             (f2cl-lib:fdo (i (f2cl-lib:int-add start 1)

```

```

        (f2cl-lib:int-add i 1))
      (> i endd) nil)
    (tagbody
      (f2cl-lib:fdo (j i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
        (> j (f2cl-lib:int-add start 1)) nil)
      (tagbody
        (cond
          ((> (f2cl-lib:fref d (j) ((1 *)))
              (f2cl-lib:fref d
                            ((f2cl-lib:int-add j
                                                  (f2cl-lib:int-sub
                                                    1)))
                            ((1 *))))
            (setf dmnmx
                  (f2cl-lib:fref d-%data%
                                (j)
                                ((1 *))
                                d-%offset%))
            (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
                  (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-sub j 1))
                                ((1 *))
                                d-%offset%))
            (setf (f2cl-lib:fref d-%data%
                                ((f2cl-lib:int-sub j 1))
                                ((1 *))
                                d-%offset%)
                  dmnmx))
          (t
            (go label30))))))
label30)))
(t
 (f2cl-lib:fdo (i (f2cl-lib:int-add start 1)
                (f2cl-lib:int-add i 1))
  (> i endd) nil)
 (tagbody
  (f2cl-lib:fdo (j i (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j (f2cl-lib:int-add start 1)) nil)
  (tagbody
    (cond
      ((< (f2cl-lib:fref d (j) ((1 *)))
          (f2cl-lib:fref d
                        ((f2cl-lib:int-add j
                                          (f2cl-lib:int-sub
                                            1)))
                        ((1 *))))
        (setf dmnmx
              (f2cl-lib:fref d-%data%
                            (j)
                            ((1 *))
                            d-%offset%))
        (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
              (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-sub j 1))
                            ((1 *))
                            d-%offset%))
        (setf (f2cl-lib:fref d-%data%
                            ((f2cl-lib:int-sub j 1))
                            ((1 *))
                            d-%offset%)
              dmnmx))
      (t
        (go label30))))))
label30)))

```

```

                                d-%offset%))
      (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%)
            (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-sub j 1))
                          ((1 *))
                          d-%offset%))
      (setf (f2cl-lib:fref d-%data%
                          ((f2cl-lib:int-sub j 1))
                          ((1 *))
                          d-%offset%)
            dmnmx))
    (t
     (go label50))))
label50))))
((> (f2cl-lib:int-add endd (f2cl-lib:int-sub start)) select)
 (setf d1 (f2cl-lib:fref d-%data% (start) ((1 *)) d-%offset%))
 (setf d2 (f2cl-lib:fref d-%data% (endd) ((1 *)) d-%offset%))
 (setf i (the fixnum (truncate (+ start endd) 2)))
 (setf d3 (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
 (cond
  ((< d1 d2)
   (cond
    ((< d3 d1)
     (setf dmnmx d1))
    ((< d3 d2)
     (setf dmnmx d3))
    (t
     (setf dmnmx d2))))
  (t
   (cond
    ((< d3 d2)
     (setf dmnmx d2))
    ((< d3 d1)
     (setf dmnmx d3))
    (t
     (setf dmnmx d1))))))
(cond
 (= dir 0)
 (tagbody
  (setf i (f2cl-lib:int-sub start 1))
  (setf j (f2cl-lib:int-add endd 1))
label60
  (setf j (f2cl-lib:int-sub j 1))
  (if (< (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) dmnmx)
      (go label60))
label80
  (setf i (f2cl-lib:int-add i 1))
  (if (> (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) dmnmx)
      (go label80))
  (cond

```

```

((< i j)
  (setf tmp (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
        (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
  (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) tmp)
  (go label60)))
(cond
 (> (f2cl-lib:int-add j (f2cl-lib:int-sub start))
  (f2cl-lib:int-add endd
    (f2cl-lib:int-sub j)
    (f2cl-lib:int-sub 1)))
 (setf stkpnt (f2cl-lib:int-add stkpnt 1))
 (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
 (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j)
 (setf stkpnt (f2cl-lib:int-add stkpnt 1))
 (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
       (f2cl-lib:int-add j 1))
 (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd))
(t
 (setf stkpnt (f2cl-lib:int-add stkpnt 1))
 (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
       (f2cl-lib:int-add j 1))
 (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd)
 (setf stkpnt (f2cl-lib:int-add stkpnt 1))
 (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
 (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j))))
(t
 (tagbody
  (setf i (f2cl-lib:int-sub start 1))
  (setf j (f2cl-lib:int-add endd 1))

label90
  (setf j (f2cl-lib:int-sub j 1))
  (if (> (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) dmmmx)
    (go label90))

label110
  (setf i (f2cl-lib:int-add i 1))
  (if (< (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%) dmmmx)
    (go label110))
  (cond
   ((< i j)
    (setf tmp (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref d-%data% (i) ((1 *)) d-%offset%)
          (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%))
    (setf (f2cl-lib:fref d-%data% (j) ((1 *)) d-%offset%) tmp)
    (go label90)))
   (cond
    (> (f2cl-lib:int-add j (f2cl-lib:int-sub start))
      (f2cl-lib:int-add endd
        (f2cl-lib:int-sub j)
        (f2cl-lib:int-sub 1)))
    (f2cl-lib:int-sub j)
    (f2cl-lib:int-sub 1)))

```

```

      (setf stkpnt (f2cl-lib:int-add stkpnt 1))
      (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
      (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) j)
      (setf stkpnt (f2cl-lib:int-add stkpnt 1))
      (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
            (f2cl-lib:int-add j 1))
      (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd))
    (t
      (setf stkpnt (f2cl-lib:int-add stkpnt 1))
      (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32)))
            (f2cl-lib:int-add j 1))
      (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32))) endd))
      (setf stkpnt (f2cl-lib:int-add stkpnt 1))
      (setf (f2cl-lib:fref stack (1 stkpnt) ((1 2) (1 32))) start)
      (setf (f2cl-lib:fref stack (2 stkpnt) ((1 2) (1 32)))
            j))))))
    (if (> stkpnt 0) (go label10))
  end_label
  (return (values nil nil nil info))))))

```

dllassq LAPACK

— dllassq.input —

```

)set break resume
)sys rm -f dllassq.output
)spool dllassq.output
)set message test on
)set message auto off
)clear all

```

```

)spool
)lisp (bye)

```

— dllassq.help —

```

=====
dllassq examples
=====
=====

```

Man Page Details

=====

NAME

DLASSQ - the values scl and smsq such that $(scl**2)*smsq = x(1)**2 + \dots + x(n)**2 + (scale**2)*sumsq$,

SYNOPSIS

SUBROUTINE DLASSQ(N, X, INCX, SCALE, SUMSQ)

INTEGER INCX, N

DOUBLE PRECISION SCALE, SUMSQ

DOUBLE PRECISION X(*)

Purpose

=====

DLASSQ returns the values scl and smsq such that

$$(scl**2)*smsq = x(1)**2 + \dots + x(n)**2 + (scale**2)*sumsq,$$

where $x(i) = X(1 + (i - 1)*INCX)$. The value of sumsq is assumed to be non-negative and scl returns the value

$$scl = \max(scale, \text{abs}(x(i)))$$

scale and sumsq must be supplied in SCALE and SUMSQ and scl and smsq are overwritten on SCALE and SUMSQ respectively.

The routine makes only one pass through the vector x.

Arguments

=====

N (input) INTEGER

The number of elements to be used from the vector X.

X (input) DOUBLE PRECISION array, dimension (N)

The vector for which a scaled sum of squares is computed.

$$x(i) = X(1 + (i - 1)*INCX), 1 \leq i \leq n.$$

INCX (input) INTEGER

The increment between successive values of the vector X.
INCX > 0.

SCALE (input/output) DOUBLE PRECISION

On entry, the value scale in the equation above.

On exit, SCALE is overwritten with scl, the scaling factor

for the sum of squares.

SUMSQ (input/output) DOUBLE PRECISION
 On entry, the value `sumsq` in the equation above.
 On exit, SUMSQ is overwritten with `sumsq`, the basic sum of
 squares from which `scl` has been factored out.

— dlassq.f —

```

SUBROUTINE DLASSQ( N, X, INCX, SCALE, SUMSQ )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      INTEGER          INCX, N
      DOUBLE PRECISION SCALE, SUMSQ
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION X( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER        ( ZERO = 0.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
      INTEGER          IX
      DOUBLE PRECISION ABSXI
*
*  ..
*
*  .. Intrinsic Functions ..
      INTRINSIC        ABS
*
*  ..
*
*  .. Executable Statements ..
*
      IF( N.GT.0 ) THEN
        DO 10 IX = 1, 1 + ( N-1 )*INCX, INCX
          IF( X( IX ).NE.ZERO ) THEN
            ABSXI = ABS( X( IX ) )
            IF( SCALE.LT.ABSXI ) THEN
              SUMSQ = 1 + SUMSQ*( SCALE / ABSXI )**2
              SCALE = ABSXI
            
```

```

        ELSE
            SUMSQ = SUMSQ + ( ABSXI / SCALE )**2
        END IF
    END IF
10    CONTINUE
    END IF
    RETURN
*
*    End of DCLASSQ
*
    END

```

— LAPACK dlassq —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dlassq (n x incx scale sumsq)
    (declare (type (double-float) sumsq scale)
              (type (simple-array double-float (*)) x)
              (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
      ((x double-float x-%data% x-%offset%))
      (prog ((absxi 0.0) (ix 0))
        (declare (type (double-float) absxi) (type fixnum ix))
        (cond
          ((> n 0)
           (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
                         ((> ix
                           (f2cl-lib:int-add 1
                             (f2cl-lib:int-mul
                               (f2cl-lib:int-add n
                                (f2cl-lib:int-sub 1))
                               incx)))
                             nil)
           (tagbody
            (cond
              ((/= (f2cl-lib:fref x (ix) ((1 *))) zero)
               (setf absxi
                     (abs
                      (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%)))
              (cond
                ((< scale absxi)
                 (setf sumsq (+ 1 (* sumsq (expt (/ scale absxi) 2))))
                 (setf scale absxi))
                (t
                 (setf sumsq (+ sumsq (expt (/ absxi scale) 2))))))))))

```



```
(return (values nil nil nil scale sumsq))))))
```

dlasv2 LAPACK

— dlasv2.input —

```
)set break resume
)sys rm -f dlasv2.output
)spool dlasv2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dlasv2.help —

```
=====
dlasv2 examples
=====
```

```
=====
Man Page Details
=====
```

NAME

```
DLASV2 - the singular value decomposition of a 2-by-2 triangular matrix
[ F G ] [ O H ]
```

SYNOPSIS

```
SUBROUTINE DLASV2( F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL )
```

```
DOUBLE          PRECISION CSL, CSR, F, G, H, SNL, SNR, SSMAX, SSMIN
```

Purpose

```
=====
```

```
DLASV2 computes the singular value decomposition of a 2-by-2
triangular matrix
```

```
[ F G ]
```

$\begin{bmatrix} 0 & H \end{bmatrix}$.

On return, `abs(SSMAX)` is the larger singular value, `abs(SSMIN)` is the smaller singular value, and `(CSL,SNL)` and `(CSR,SNR)` are the left and right singular vectors for `abs(SSMAX)`, giving the decomposition

$$\begin{bmatrix} \text{CSL} & \text{SNL} \end{bmatrix} \begin{bmatrix} F & G \end{bmatrix} \begin{bmatrix} \text{CSR} & -\text{SNR} \end{bmatrix} = \begin{bmatrix} \text{SSMAX} & 0 \end{bmatrix} \\ \begin{bmatrix} -\text{SNL} & \text{CSL} \end{bmatrix} \begin{bmatrix} 0 & H \end{bmatrix} \begin{bmatrix} \text{SNR} & \text{CSR} \end{bmatrix} = \begin{bmatrix} 0 & \text{SSMIN} \end{bmatrix}.$$

Arguments

=====

F (input) DOUBLE PRECISION
The (1,1) element of the 2-by-2 matrix.

G (input) DOUBLE PRECISION
The (1,2) element of the 2-by-2 matrix.

H (input) DOUBLE PRECISION
The (2,2) element of the 2-by-2 matrix.

SSMIN (output) DOUBLE PRECISION
`abs(SSMIN)` is the smaller singular value.

SSMAX (output) DOUBLE PRECISION
`abs(SSMAX)` is the larger singular value.

SNL (output) DOUBLE PRECISION
CSL (output) DOUBLE PRECISION
The vector `(CSL, SNL)` is a unit left singular vector for the singular value `abs(SSMAX)`.

SNR (output) DOUBLE PRECISION
CSR (output) DOUBLE PRECISION
The vector `(CSR, SNR)` is a unit right singular vector for the singular value `abs(SSMAX)`.

Further Details

=====

Any input parameter may be aliased with any output parameter.

Barring over/underflow and assuming a guard digit in subtraction, all output quantities are correct to within a few units in the last place (ulps).

In IEEE arithmetic, the code works correctly if one matrix element is infinite.

Overflow will not occur unless the largest singular value itself overflows or is within a few ulps of overflow. (On machines with

partial overflow, like the Cray, overflow may occur if the largest singular value is within a factor of 2 of overflow.)

Underflow is harmless if underflow is gradual. Otherwise, results may correspond to a matrix modified by perturbations of size near the underflow threshold.

— dlasv2.f —

```

SUBROUTINE DLASV2( F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    October 31, 1992
*
*    .. Scalar Arguments ..
*    DOUBLE PRECISION    CSL, CSR, F, G, H, SNL, SNR, SSMAX, SSMIN
*    ..
*
*    =====
*
*    .. Parameters ..
*    DOUBLE PRECISION    ZERO
*    PARAMETER            ( ZERO = 0.0D0 )
*    DOUBLE PRECISION    HALF
*    PARAMETER            ( HALF = 0.5D0 )
*    DOUBLE PRECISION    ONE
*    PARAMETER            ( ONE = 1.0D0 )
*    DOUBLE PRECISION    TWO
*    PARAMETER            ( TWO = 2.0D0 )
*    DOUBLE PRECISION    FOUR
*    PARAMETER            ( FOUR = 4.0D0 )
*
*    ..
*    .. Local Scalars ..
*    LOGICAL              GASMAL, SWAP
*    INTEGER              PMAX
*    DOUBLE PRECISION     A, CLT, CRT, D, FA, FT, GA, GT, HA, HT, L, M,
*    $                    MM, R, S, SLT, SRT, T, TEMP, TSIGN, TT
*
*    ..
*    .. Intrinsic Functions ..
*    INTRINSIC            ABS, SIGN, SQRT
*
*    ..
*    .. External Functions ..
*    DOUBLE PRECISION     DLAMCH
*    EXTERNAL              DLAMCH

```

```

*      ..
*      .. Executable Statements ..
*
      FT = F
      FA = ABS( FT )
      HT = H
      HA = ABS( H )
*
*      PMAX points to the maximum absolute element of matrix
*      PMAX = 1 if F largest in absolute values
*      PMAX = 2 if G largest in absolute values
*      PMAX = 3 if H largest in absolute values
*
      PMAX = 1
      SWAP = ( HA.GT.FA )
      IF( SWAP ) THEN
         PMAX = 3
         TEMP = FT
         FT = HT
         HT = TEMP
         TEMP = FA
         FA = HA
         HA = TEMP
*
*      Now FA .ge. HA
*
      END IF
      GT = G
      GA = ABS( GT )
      IF( GA.EQ.ZERO ) THEN
*
*      Diagonal matrix
*
         SSMIN = HA
         SSMAX = FA
         CLT = ONE
         CRT = ONE
         SLT = ZERO
         SRT = ZERO
      ELSE
         GASMAL = .TRUE.
         IF( GA.GT.FA ) THEN
            PMAX = 2
            IF( ( FA / GA ).LT.DLAMCH( 'EPS' ) ) THEN
*
*      Case of very large GA
*
               GASMAL = .FALSE.
               SSMAX = GA
               IF( HA.GT.ONE ) THEN

```

```

        SSMIN = FA / ( GA / HA )
    ELSE
        SSMIN = ( FA / GA ) * HA
    END IF
    CLT = ONE
    SLT = HT / GT
    SRT = ONE
    CRT = FT / GT
    END IF
END IF
IF( GASMAL ) THEN
*
*       Normal case
*
    D = FA - HA
    IF( D.EQ.FA ) THEN
*
*       Copes with infinite F or H
*
        L = ONE
    ELSE
        L = D / FA
    END IF
*
*       Note that 0 .le. L .le. 1
*
    M = GT / FT
*
*       Note that abs(M) .le. 1/macheps
*
    T = TWO - L
*
*       Note that T .ge. 1
*
    MM = M*M
    TT = T*T
    S = SQRT( TT+MM )
*
*       Note that 1 .le. S .le. 1 + 1/macheps
*
    IF( L.EQ.ZERO ) THEN
        R = ABS( M )
    ELSE
        R = SQRT( L*L+MM )
    END IF
*
*       Note that 0 .le. R .le. 1 + 1/macheps
*
    A = HALF*( S+R )
*

```

```

*          Note that  $1 \leq A \leq 1 + \text{abs}(M)$ 
*
      SSMIN = HA / A
      SSMAX = FA*A
      IF( MM.EQ.ZERO ) THEN
*
*          Note that M is very tiny
*
      IF( L.EQ.ZERO ) THEN
        T = SIGN( TWO, FT )*SIGN( ONE, GT )
      ELSE
        T = GT / SIGN( D, FT ) + M / T
      END IF
      ELSE
        T = ( M / ( S+T )+M / ( R+L ) )*( ONE+A )
      END IF
      L = SQRT( T*T+FOUR )
      CRT = TWO / L
      SRT = T / L
      CLT = ( CRT+SRT*M ) / A
      SLT = ( HT / FT )*SRT / A
    END IF
  END IF
  IF( SWAP ) THEN
    CSL = SRT
    SNL = CRT
    CSR = SLT
    SNR = CLT
  ELSE
    CSL = CLT
    SNL = SLT
    CSR = CRT
    SNR = SRT
  END IF
*
*  Correct signs of SSMAX and SSMIN
*
  IF( PMAX.EQ.1 )
    $  TSIGN = SIGN( ONE, CSR )*SIGN( ONE, CSL )*SIGN( ONE, F )
  IF( PMAX.EQ.2 )
    $  TSIGN = SIGN( ONE, SNR )*SIGN( ONE, CSL )*SIGN( ONE, G )
  IF( PMAX.EQ.3 )
    $  TSIGN = SIGN( ONE, SNR )*SIGN( ONE, SNL )*SIGN( ONE, H )
    SSMAX = SIGN( SSMAX, TSIGN )
    SSMIN = SIGN( SSMIN, TSIGN*SIGN( ONE, F )*SIGN( ONE, H ) )
    RETURN
*
*  End of DLASV2
*
END

```

— LAPACK dlasv2 —

```

(let* ((zero 0.0) (half 0.5) (one 1.0) (two 2.0) (four 4.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 0.5 0.5) half)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 4.0 4.0) four))
  (defun dlasv2 (f g h ssmin ssmax snr csr snl csl)
    (declare (type (double-float) csl snl csr snr ssmax ssmin h g f))
    (prog ((a 0.0) (clt 0.0) (crt 0.0) (d 0.0) (fa 0.0) (ft 0.0) (ga 0.0)
           (gt 0.0) (ha 0.0) (ht 0.0) (l 0.0) (m 0.0) (mm 0.0) (r 0.0) (s 0.0)
           (slt 0.0) (srt 0.0) (t$ 0.0) (temp 0.0) (tsign 0.0) (tt 0.0)
           (pmax 0) (gasmal nil) (swap nil))
      (declare (type (double-float) a clt crt d fa ft ga gt ha ht l m mm r s
                    slt srt t$ temp tsign tt)
                (type fixnum pmax)
                (type (member t nil) gasmal swap))
      (setf ft f)
      (setf fa (abs ft))
      (setf ht h)
      (setf ha (abs h))
      (setf pmax 1)
      (setf swap (> ha fa))
      (cond
        (swap
         (setf pmax 3)
         (setf temp ft)
         (setf ft ht)
         (setf ht temp)
         (setf temp fa)
         (setf fa ha)
         (setf ha temp)))
      (setf gt g)
      (setf ga (abs gt))
      (cond
        ((= ga zero)
         (setf ssmin ha)
         (setf ssmax fa)
         (setf clt one)
         (setf crt one)
         (setf slt zero)
         (setf srt zero))
      (t
       (setf gasmal t)

```

```

(cond
  (> ga fa)
  (setf pmax 2)
  (cond
    (< (f2cl-lib:f2cl/ fa ga) (dlamch "EPS"))
    (setf gasmal nil)
    (setf ssmax ga)
    (cond
      (> ha one)
      (setf ssmin (/ fa (/ ga ha))))
    (t
      (setf ssmin (* (/ fa ga) ha))))
    (setf clt one)
    (setf slt (/ ht gt))
    (setf srt one)
    (setf crt (/ ft gt))))))
(cond
  (gasmal
    (setf d (- fa ha))
    (cond
      (= d fa)
      (setf l one))
    (t
      (setf l (/ d fa))))
    (setf m (/ gt ft))
    (setf t$ (- two l))
    (setf mm (* m m))
    (setf tt (* t$ t$))
    (setf s (f2cl-lib:fsqrt (+ tt mm)))
    (cond
      (= l zero)
      (setf r (abs m)))
    (t
      (setf r (f2cl-lib:fsqrt (+ (* l l) mm))))
    (setf a (* half (+ s r)))
    (setf ssmin (/ ha a))
    (setf ssmax (* fa a))
    (cond
      (= mm zero)
      (cond
        (= l zero)
        (setf t$ (* (f2cl-lib:sign two ft) (f2cl-lib:sign one gt))))
      (t
        (setf t$ (+ (/ gt (f2cl-lib:sign d ft)) (/ m t$))))))
    (t
      (setf t$ (* (+ (/ m (+ s t$)) (/ m (+ r l))) (+ one a))))
    (setf l (f2cl-lib:fsqrt (+ (* t$ t$) four)))
    (setf crt (/ two l))
    (setf srt (/ t$ l))
    (setf clt (/ (+ crt (* srt m)) a))

```



```

        (setf slt (/ (* (/ ht ft) srt) a))))))
(cond
  (swap
    (setf csl srt)
    (setf snl crt)
    (setf csr slt)
    (setf snr clt))
  (t
    (setf csl clt)
    (setf snl slt)
    (setf csr crt)
    (setf snr srt)))
(if (= pmax 1)
  (setf tsign
    (* (f2cl-lib:sign one csr)
       (f2cl-lib:sign one csl)
       (f2cl-lib:sign one f))))
(if (= pmax 2)
  (setf tsign
    (* (f2cl-lib:sign one snr)
       (f2cl-lib:sign one csl)
       (f2cl-lib:sign one g))))
(if (= pmax 3)
  (setf tsign
    (* (f2cl-lib:sign one snr)
       (f2cl-lib:sign one snl)
       (f2cl-lib:sign one h))))
(setf ssmax (f2cl-lib:sign ssmax tsign))
(setf ssmin
  (f2cl-lib:sign ssmin
    (* tsign
      (f2cl-lib:sign one f)
      (f2cl-lib:sign one h))))
(return (values nil nil nil ssmin ssmax snr csr snl csl))))

```

dlaswp LAPACK

— dlaswp.input —

```

)set break resume
)sys rm -f dlaswp.output
)spool dlaswp.output
)set message test on
)set message auto off

```

```

)clear all

)spool
)lisp (bye)

```

— dlaswp.help —

```

=====
dlaswp examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASWP - a series of row interchanges on the matrix A

SYNOPSIS

```

SUBROUTINE DLASWP( N, A, LDA, K1, K2, IPIV, INCX )

```

```

      INTEGER      INCX, K1, K2, LDA, N

```

```

      INTEGER      IPIV( * )

```

```

      DOUBLE      PRECISION A( LDA, * )

```

Purpose

=====

DLASWP performs a series of row interchanges on the matrix A.
One row interchange is initiated for each of rows K1 through K2 of A.

Arguments

=====

N (input) INTEGER
 The number of columns of the matrix A.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the matrix of column dimension N to which the row
 interchanges will be applied.
 On exit, the permuted matrix.

LDA (input) INTEGER
 The leading dimension of the array A.

K1 (input) INTEGER
 The first element of IPIV for which a row interchange will
 be done.

K2 (input) INTEGER
 The last element of IPIV for which a row interchange will
 be done.

IPIV (input) INTEGER array, dimension (M*abs(INCX))
 The vector of pivot indices. Only the elements in positions
 K1 through K2 of IPIV are accessed.
 IPIV(K) = L implies rows K and L are to be interchanged.

INCX (input) INTEGER
 The increment between successive values of IPIV. If IPIV
 is negative, the pivots are applied in reverse order.

Further Details
 =====

Modified by
 R. C. Whaley, Computer Science Dept., Univ. of Tenn., Knoxville, USA

— dlaswp.f —

```

SUBROUTINE DLASWP( N, A, LDA, K1, K2, IPIV, INCX )
*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
INTEGER          INCX, K1, K2, LDA, N
*
*  ..
*
*  .. Array Arguments ..
INTEGER          IPIV( * )
DOUBLE PRECISION A( LDA, * )
*
*  ..
*
*  =====
*
*  .. Local Scalars ..
INTEGER          I, I1, I2, INC, IP, IX, IX0, J, K, N32
DOUBLE PRECISION TEMP
*
*  ..

```

```

*      .. Executable Statements ..
*
*      Interchange row I with row IPIV(I) for each of rows K1 through K2.
*
      IF( INCX.GT.0 ) THEN
        IX0 = K1
        I1 = K1
        I2 = K2
        INC = 1
      ELSE IF( INCX.LT.0 ) THEN
        IX0 = 1 + ( 1-K2 )*INCX
        I1 = K2
        I2 = K1
        INC = -1
      ELSE
        RETURN
      END IF
*
      N32 = ( N / 32 )*32
      IF( N32.NE.0 ) THEN
        DO 30 J = 1, N32, 32
          IX = IX0
          DO 20 I = I1, I2, INC
            IP = IPIV( IX )
            IF( IP.NE.I ) THEN
              DO 10 K = J, J + 31
                TEMP = A( I, K )
                A( I, K ) = A( IP, K )
                A( IP, K ) = TEMP
10              CONTINUE
            END IF
            IX = IX + INCX
          20 CONTINUE
        30 CONTINUE
      END IF
      IF( N32.NE.N ) THEN
        N32 = N32 + 1
        IX = IX0
        DO 50 I = I1, I2, INC
          IP = IPIV( IX )
          IF( IP.NE.I ) THEN
            DO 40 K = N32, N
              TEMP = A( I, K )
              A( I, K ) = A( IP, K )
              A( IP, K ) = TEMP
40            CONTINUE
          END IF
          IX = IX + INCX
        50 CONTINUE
      END IF

```

```

*
*      RETURN
*
*      End of DLASWP
*
*      END

```

— LAPACK dlaswp —

```

(defun dlaswp (n a lda k1 k2 ipiv incx)
  (declare (type (simple-array fixnum (*)) ipiv)
            (type (simple-array double-float (*)) a)
            (type fixnum incx k2 k1 lda n))
  (f2cl-lib:with-multi-array-data
    ((a double-float a-%data% a-%offset%)
     (ipiv fixnum ipiv-%data% ipiv-%offset%))
    (prog ((temp 0.0) (i 0) (i1 0) (i2 0) (inc 0) (ip 0) (ix 0) (ix0 0) (j 0)
           (k 0) (n32 0))
      (declare (type fixnum n32 k j ix0 ix ip inc i2 i1 i)
                (type (double-float) temp))
      (cond
        ((> incx 0)
         (setf ix0 k1)
         (setf i1 k1)
         (setf i2 k2)
         (setf inc 1))
        ((< incx 0)
         (setf ix0
                (f2cl-lib:int-add 1
                                   (f2cl-lib:int-mul (f2cl-lib:int-sub 1 k2)
                                                       incx))))
        (t
         (setf i1 k2)
         (setf i2 k1)
         (setf inc -1))
        (t
         (go end_label))))
  (setf n32 (* (the fixnum (truncate n 32)) 32))
  (cond
    ((/= n32 0)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 32))
                   ((> j n32) nil)
     (tagbody
      (setf ix ix0)
      (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i inc))
                    ((> i i2) nil)
      (tagbody

```

```

      (setf ip
        (f2cl-lib:fref ipiv-%data%
          (ix)
          ((1 *))
          ipiv-%offset%))
      (cond
        ((/= ip i)
         (f2cl-lib:fdo (k j (f2cl-lib:int-add k 1))
           (> k (f2cl-lib:int-add j 31)) nil)
         (tagbody
           (setf temp
             (f2cl-lib:fref a-%data%
              (i k)
              ((1 lda) (1 *))
              a-%offset%))
             (setf (f2cl-lib:fref a-%data%
              (i k)
              ((1 lda) (1 *))
              a-%offset%)
              (f2cl-lib:fref a-%data%
              (ip k)
              ((1 lda) (1 *))
              a-%offset%))
             (setf (f2cl-lib:fref a-%data%
              (ip k)
              ((1 lda) (1 *))
              a-%offset%)
              temp))))
        (setf ix (f2cl-lib:int-add ix incx))))))
      (cond
        ((/= n32 n)
         (setf n32 (f2cl-lib:int-add n32 1))
         (setf ix ix0)
         (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i inc))
           (> i i2) nil)
         (tagbody
           (setf ip (f2cl-lib:fref ipiv-%data% (ix) ((1 *)) ipiv-%offset%))
           (cond
             ((/= ip i)
              (f2cl-lib:fdo (k n32 (f2cl-lib:int-add k 1))
                (> k n) nil)
              (tagbody
                (setf temp
                  (f2cl-lib:fref a-%data%
                   (i k)
                   ((1 lda) (1 *))
                   a-%offset%))
                  (setf (f2cl-lib:fref a-%data%
                   (i k)
                   ((1 lda) (1 *))

```

```

                                a-%offset%)
      (f2cl-lib:fref a-%data%
        (ip k)
        ((1 lda) (1 *))
        a-%offset%))
    (setf (f2cl-lib:fref a-%data%
      (ip k)
      ((1 lda) (1 *))
      a-%offset%)
      temp))))
    (setf ix (f2cl-lib:int-add ix incx))))))
end_label
  (return (values nil nil nil nil nil nil nil))))

```

dlasy2 LAPACK

— dlasy2.input —

```

)set break resume
)sys rm -f dlasy2.output
)spool dlasy2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dlasy2.help —

```

=====
dlasy2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DLASY2 - for the N1 by N2 matrix X, $1 \leq N1, N2 \leq 2$, in $op(TL)*X +$
 $ISGN*X*op(TR) = SCALE*B$,

SYNOPSIS

```
SUBROUTINE DLASY2( LTRANL, LTRANR, ISGN, N1, N2, TL, LDTL, TR, LDTR, B,
                  LDB, SCALE, X, LDX, XNORM, INFO )
```

```
LOGICAL          LTRANL, LTRANR

INTEGER          INFO, ISGN, LDB, LDTL, LDTR, LDX, N1, N2

DOUBLE          PRECISION SCALE, XNORM

DOUBLE          PRECISION B( LDB, * ), TL( LDTL, * ), TR( LDTR, * ),
                  X( LDX, * )
```

Purpose

```
=====
```

DLASY2 solves for the $N1$ by $N2$ matrix X , $1 \leq N1, N2 \leq 2$, in

$$\text{op}(\text{TL}) * X + \text{ISGN} * X * \text{op}(\text{TR}) = \text{SCALE} * B,$$

where TL is $N1$ by $N1$, TR is $N2$ by $N2$, B is $N1$ by $N2$, and $\text{ISGN} = 1$ or -1 . $\text{op}(T) = T$ or T' , where T' denotes the transpose of T .

Arguments

```
=====
```

```
LTRANL  (input) LOGICAL
On entry, LTRANL specifies the op(TL):
    = .FALSE., op(TL) = TL,
    = .TRUE.,  op(TL) = TL'.

LTRANR  (input) LOGICAL
On entry, LTRANR specifies the op(TR):
    = .FALSE., op(TR) = TR,
    = .TRUE.,  op(TR) = TR'.

ISGN     (input) INTEGER
On entry, ISGN specifies the sign of the equation
as described before. ISGN may only be 1 or -1.

N1       (input) INTEGER
On entry, N1 specifies the order of matrix TL.
N1 may only be 0, 1 or 2.

N2       (input) INTEGER
On entry, N2 specifies the order of matrix TR.
N2 may only be 0, 1 or 2.

TL       (input) DOUBLE PRECISION array, dimension (LDTL,2)
```


On entry, TL contains an N1 by N1 matrix.

LDTL (input) INTEGER
The leading dimension of the matrix TL. LDTL \geq max(1,N1).

TR (input) DOUBLE PRECISION array, dimension (LDTR,2)
On entry, TR contains an N2 by N2 matrix.

LDTR (input) INTEGER
The leading dimension of the matrix TR. LDTR \geq max(1,N2).

B (input) DOUBLE PRECISION array, dimension (LDB,2)
On entry, the N1 by N2 matrix B contains the right-hand side of the equation.

LDB (input) INTEGER
The leading dimension of the matrix B. LDB \geq max(1,N1).

SCALE (output) DOUBLE PRECISION
On exit, SCALE contains the scale factor. SCALE is chosen less than or equal to 1 to prevent the solution overflowing.

X (output) DOUBLE PRECISION array, dimension (LDX,2)
On exit, X contains the N1 by N2 solution.

LDX (input) INTEGER
The leading dimension of the matrix X. LDX \geq max(1,N1).

XNORM (output) DOUBLE PRECISION
On exit, XNORM is the infinity-norm of the solution.

INFO (output) INTEGER
On exit, INFO is set to
0: successful exit.
1: TL and TR have too close eigenvalues, so TL or TR is perturbed to get a nonsingular equation.
NOTE: In the interests of speed, this routine does not check the inputs for errors.

— dlasy2.f —

```

SUBROUTINE DLASY2( LTRANL, LTRANR, ISGN, N1, N2, TL, LDTL, TR,
$                LDTR, B, LDB, SCALE, X, LDX, XNORM, INFO )
*
* -- LAPACK auxiliary routine (version 3.0) --
* Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
```

```

*      Courant Institute, Argonne National Lab, and Rice University
*      October 31, 1992
*
*      .. Scalar Arguments ..
LOGICAL          LTRANL, LTRANR
INTEGER          INFO, ISGN, LDB, LDTL, LDTR, LDX, N1, N2
DOUBLE PRECISION SCALE, XNORM
*
*      ..
*      .. Array Arguments ..
DOUBLE PRECISION B( LDB, * ), TL( LDTL, * ), TR( LDTR, * ),
$                X( LDX, * )
*      ..
*
*      =====
*
*      .. Parameters ..
DOUBLE PRECISION ZERO, ONE
PARAMETER        ( ZERO = 0.0D+0, ONE = 1.0D+0 )
DOUBLE PRECISION TWO, HALF, EIGHT
PARAMETER        ( TWO = 2.0D+0, HALF = 0.5D+0, EIGHT = 8.0D+0 )
*
*      ..
*      .. Local Scalars ..
LOGICAL          BSWAP, XSWAP
INTEGER          I, IP, IPIV, IPSV, J, JP, JPSV, K
DOUBLE PRECISION BET, EPS, GAM, L21, SGN, SMLNUM, TAU1,
$                TEMP, U11, U12, U22, XMAX
*
*      ..
*      .. Local Arrays ..
LOGICAL          BSWPIV( 4 ), XSWPIV( 4 )
INTEGER          JPIV( 4 ), LOCL21( 4 ), LOCU12( 4 ),
$                LOCU22( 4 )
DOUBLE PRECISION BTMP( 4 ), T16( 4, 4 ), TMP( 4 ), X2( 2 )
*
*      ..
*      .. External Functions ..
INTEGER          IDAMAX
DOUBLE PRECISION DLAMCH
EXTERNAL         IDAMAX, DLAMCH
*
*      ..
*      .. External Subroutines ..
EXTERNAL         DCOPY, DSWAP
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC        ABS, MAX
*
*      ..
*      .. Data statements ..
DATA             LOCU12 / 3, 4, 1, 2 / , LOCL21 / 2, 1, 4, 3 / ,
$                LOCU22 / 4, 3, 2, 1 /
DATA             XSWPIV / .FALSE., .FALSE., .TRUE., .TRUE. /
DATA             BSWPIV / .FALSE., .TRUE., .FALSE., .TRUE. /
*      ..

```

```

*      .. Executable Statements ..
*
*      Do not check the input parameters for errors
*
*      INFO = 0
*
*      Quick return if possible
*
*      IF( N1.EQ.0 .OR. N2.EQ.0 )
*      $   RETURN
*
*      Set constants to control overflow
*
*      EPS = DLAMCH( 'P' )
*      SMLNUM = DLAMCH( 'S' ) / EPS
*      SGN = ISGN
*
*      K = N1 + N1 + N2 - 2
*      GO TO ( 10, 20, 30, 50 )K
*
*      1 by 1: TL11*X + SGN*X*TR11 = B11
*
*      10 CONTINUE
*      TAU1 = TL( 1, 1 ) + SGN*TR( 1, 1 )
*      BET = ABS( TAU1 )
*      IF( BET.LE.SMLNUM ) THEN
*          TAU1 = SMLNUM
*          BET = SMLNUM
*          INFO = 1
*      END IF
*
*      SCALE = ONE
*      GAM = ABS( B( 1, 1 ) )
*      IF( SMLNUM*GAM.GT.BET )
*      $   SCALE = ONE / GAM
*
*      X( 1, 1 ) = ( B( 1, 1 )*SCALE ) / TAU1
*      XNORM = ABS( X( 1, 1 ) )
*      RETURN
*
*      1 by 2:
*      TL11*[X11 X12] + ISGN*[X11 X12]*op[TR11 TR12] = [B11 B12]
*                                              [TR21 TR22]
*
*      20 CONTINUE
*
*      SMIN = MAX( EPS*MAX( ABS( TL( 1, 1 ) ), ABS( TR( 1, 1 ) ),
*      $          ABS( TR( 1, 2 ) ), ABS( TR( 2, 1 ) ), ABS( TR( 2, 2 ) ) ),
*      $          SMLNUM )
*      TMP( 1 ) = TL( 1, 1 ) + SGN*TR( 1, 1 )

```

```

      TMP( 4 ) = TL( 1, 1 ) + SGN*TR( 2, 2 )
      IF( LTRANR ) THEN
         TMP( 2 ) = SGN*TR( 2, 1 )
         TMP( 3 ) = SGN*TR( 1, 2 )
      ELSE
         TMP( 2 ) = SGN*TR( 1, 2 )
         TMP( 3 ) = SGN*TR( 2, 1 )
      END IF
      BTMP( 1 ) = B( 1, 1 )
      BTMP( 2 ) = B( 1, 2 )
      GO TO 40

*
*   2 by 1:
*   op[TL11 TL12]*[X11] + ISGN* [X11]*TR11 = [B11]
*   [TL21 TL22] [X21]          [X21]      [B21]
*
30 CONTINUE
      SMIN = MAX( EPS*MAX( ABS( TR( 1, 1 ) ), ABS( TL( 1, 1 ) ),
$      ABS( TL( 1, 2 ) ), ABS( TL( 2, 1 ) ), ABS( TL( 2, 2 ) ) ),
$      SMLNUM )
      TMP( 1 ) = TL( 1, 1 ) + SGN*TR( 1, 1 )
      TMP( 4 ) = TL( 2, 2 ) + SGN*TR( 1, 1 )
      IF( LTRANL ) THEN
         TMP( 2 ) = TL( 1, 2 )
         TMP( 3 ) = TL( 2, 1 )
      ELSE
         TMP( 2 ) = TL( 2, 1 )
         TMP( 3 ) = TL( 1, 2 )
      END IF
      BTMP( 1 ) = B( 1, 1 )
      BTMP( 2 ) = B( 2, 1 )
40 CONTINUE

*
*   Solve 2 by 2 system using complete pivoting.
*   Set pivots less than SMIN to SMIN.
*
      IPIV = IDAMAX( 4, TMP, 1 )
      U11 = TMP( IPIV )
      IF( ABS( U11 ).LE.SMIN ) THEN
         INFO = 1
         U11 = SMIN
      END IF
      U12 = TMP( LOC12( IPIV ) )
      L21 = TMP( LOCL21( IPIV ) ) / U11
      U22 = TMP( LOC22( IPIV ) ) - U12*L21
      XSWAP = XSWPIV( IPIV )
      BSWAP = BSWPIV( IPIV )
      IF( ABS( U22 ).LE.SMIN ) THEN
         INFO = 1
         U22 = SMIN

```

```

      END IF
      IF( BSWAP ) THEN
        TEMP = BTMP( 2 )
        BTMP( 2 ) = BTMP( 1 ) - L21*TEMP
        BTMP( 1 ) = TEMP
      ELSE
        BTMP( 2 ) = BTMP( 2 ) - L21*BTMP( 1 )
      END IF
      SCALE = ONE
      IF( ( TWO*SMLNUM )*ABS( BTMP( 2 ) ).GT.ABS( U22 ) .OR.
$      ( TWO*SMLNUM )*ABS( BTMP( 1 ) ).GT.ABS( U11 ) ) THEN
        SCALE = HALF / MAX( ABS( BTMP( 1 ) ), ABS( BTMP( 2 ) ) )
        BTMP( 1 ) = BTMP( 1 )*SCALE
        BTMP( 2 ) = BTMP( 2 )*SCALE
      END IF
      X2( 2 ) = BTMP( 2 ) / U22
      X2( 1 ) = BTMP( 1 ) / U11 - ( U12 / U11 )*X2( 2 )
      IF( XSWAP ) THEN
        TEMP = X2( 2 )
        X2( 2 ) = X2( 1 )
        X2( 1 ) = TEMP
      END IF
      X( 1, 1 ) = X2( 1 )
      IF( N1.EQ.1 ) THEN
        X( 1, 2 ) = X2( 2 )
        XNORM = ABS( X( 1, 1 ) ) + ABS( X( 1, 2 ) )
      ELSE
        X( 2, 1 ) = X2( 2 )
        XNORM = MAX( ABS( X( 1, 1 ) ), ABS( X( 2, 1 ) ) )
      END IF
      RETURN
*
*
*   2 by 2:
*   op[TL11 TL12]*[X11 X12] +ISGN* [X11 X12]*op[TR11 TR12] = [B11 B12]
*   [TL21 TL22] [X21 X22]          [X21 X22]   [TR21 TR22]   [B21 B22]
*
*   Solve equivalent 4 by 4 system using complete pivoting.
*   Set pivots less than SMIN to SMIN.
*
50 CONTINUE
      SMIN = MAX( ABS( TR( 1, 1 ) ), ABS( TR( 1, 2 ) ),
$      ABS( TR( 2, 1 ) ), ABS( TR( 2, 2 ) ) )
      SMIN = MAX( SMIN, ABS( TL( 1, 1 ) ), ABS( TL( 1, 2 ) ),
$      ABS( TL( 2, 1 ) ), ABS( TL( 2, 2 ) ) )
      SMIN = MAX( EPS*SMIN, SMLNUM )
      BTMP( 1 ) = ZERO
      CALL DCOPY( 16, BTMP, 0, T16, 1 )
      T16( 1, 1 ) = TL( 1, 1 ) + SGN*TR( 1, 1 )
      T16( 2, 2 ) = TL( 2, 2 ) + SGN*TR( 1, 1 )
      T16( 3, 3 ) = TL( 1, 1 ) + SGN*TR( 2, 2 )

```

```

T16( 4, 4 ) = TL( 2, 2 ) + SGN*TR( 2, 2 )
IF( LTRANL ) THEN
    T16( 1, 2 ) = TL( 2, 1 )
    T16( 2, 1 ) = TL( 1, 2 )
    T16( 3, 4 ) = TL( 2, 1 )
    T16( 4, 3 ) = TL( 1, 2 )
ELSE
    T16( 1, 2 ) = TL( 1, 2 )
    T16( 2, 1 ) = TL( 2, 1 )
    T16( 3, 4 ) = TL( 1, 2 )
    T16( 4, 3 ) = TL( 2, 1 )
END IF
IF( LTRANR ) THEN
    T16( 1, 3 ) = SGN*TR( 1, 2 )
    T16( 2, 4 ) = SGN*TR( 1, 2 )
    T16( 3, 1 ) = SGN*TR( 2, 1 )
    T16( 4, 2 ) = SGN*TR( 2, 1 )
ELSE
    T16( 1, 3 ) = SGN*TR( 2, 1 )
    T16( 2, 4 ) = SGN*TR( 2, 1 )
    T16( 3, 1 ) = SGN*TR( 1, 2 )
    T16( 4, 2 ) = SGN*TR( 1, 2 )
END IF
BTMP( 1 ) = B( 1, 1 )
BTMP( 2 ) = B( 2, 1 )
BTMP( 3 ) = B( 1, 2 )
BTMP( 4 ) = B( 2, 2 )
*
* Perform elimination
*
DO 100 I = 1, 3
    XMAX = ZERO
    DO 70 IP = I, 4
        DO 60 JP = I, 4
            IF( ABS( T16( IP, JP ) ).GE.XMAX ) THEN
                XMAX = ABS( T16( IP, JP ) )
                IPSV = IP
                JPSV = JP
            END IF
        DO 60 CONTINUE
    DO 70 CONTINUE
    IF( IPSV.NE.I ) THEN
        CALL DSWAP( 4, T16( IPSV, 1 ), 4, T16( I, 1 ), 4 )
        TEMP = BTMP( I )
        BTMP( I ) = BTMP( IPSV )
        BTMP( IPSV ) = TEMP
    END IF
    IF( JPSV.NE.I )
$      CALL DSWAP( 4, T16( 1, JPSV ), 1, T16( 1, I ), 1 )
    JPIV( I ) = JPSV

```

```

      IF( ABS( T16( I, I ) ).LT.SMIN ) THEN
        INFO = 1
        T16( I, I ) = SMIN
      END IF
      DO 90 J = I + 1, 4
        T16( J, I ) = T16( J, I ) / T16( I, I )
        BTMP( J ) = BTMP( J ) - T16( J, I )*BTMP( I )
      DO 80 K = I + 1, 4
        T16( J, K ) = T16( J, K ) - T16( J, I )*T16( I, K )
80      CONTINUE
90      CONTINUE
100 CONTINUE
      IF( ABS( T16( 4, 4 ) ).LT.SMIN )
$      T16( 4, 4 ) = SMIN
      SCALE = ONE
      IF( ( EIGHT*SMLNUM )*ABS( BTMP( 1 ) ).GT.ABS( T16( 1, 1 ) ) .OR.
$      ( EIGHT*SMLNUM )*ABS( BTMP( 2 ) ).GT.ABS( T16( 2, 2 ) ) .OR.
$      ( EIGHT*SMLNUM )*ABS( BTMP( 3 ) ).GT.ABS( T16( 3, 3 ) ) .OR.
$      ( EIGHT*SMLNUM )*ABS( BTMP( 4 ) ).GT.ABS( T16( 4, 4 ) ) ) THEN
        SCALE = ( ONE / EIGHT ) / MAX( ABS( BTMP( 1 ) ),
$      ABS( BTMP( 2 ) ), ABS( BTMP( 3 ) ), ABS( BTMP( 4 ) ) )
        BTMP( 1 ) = BTMP( 1 )*SCALE
        BTMP( 2 ) = BTMP( 2 )*SCALE
        BTMP( 3 ) = BTMP( 3 )*SCALE
        BTMP( 4 ) = BTMP( 4 )*SCALE
      END IF
      DO 120 I = 1, 4
        K = 5 - I
        TEMP = ONE / T16( K, K )
        TMP( K ) = BTMP( K )*TEMP
      DO 110 J = K + 1, 4
        TMP( K ) = TMP( K ) - ( TEMP*T16( K, J ) )*TMP( J )
110      CONTINUE
120 CONTINUE
      DO 130 I = 1, 3
        IF( JPIV( 4-I ).NE.4-I ) THEN
          TEMP = TMP( 4-I )
          TMP( 4-I ) = TMP( JPIV( 4-I ) )
          TMP( JPIV( 4-I ) ) = TEMP
        END IF
130 CONTINUE
      X( 1, 1 ) = TMP( 1 )
      X( 2, 1 ) = TMP( 2 )
      X( 1, 2 ) = TMP( 3 )
      X( 2, 2 ) = TMP( 4 )
      XNORM = MAX( ABS( TMP( 1 ) )+ABS( TMP( 3 ) ),
$      ABS( TMP( 2 ) )+ABS( TMP( 4 ) ) )
      RETURN
*
*      End of DLASY2

```

```
*
END
```

— LAPACK dlasy2 —

```
(let* ((zero 0.0) (one 1.0) (two 2.0) (half 0.5) (eight 8.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two)
            (type (double-float 0.5 0.5) half)
            (type (double-float 8.0 8.0) eight))
  (let ((locu12
        (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(3 4 1 2)))
        (loc121
        (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(2 1 4 3)))
        (locu22
        (make-array 4
                     :element-type 'fixnum
                     :initial-contents '(4 3 2 1)))
        (xswpiv
        (make-array 4 :element-type 't :initial-contents '(nil nil t t)))
        (bswpiv
        (make-array 4 :element-type 't :initial-contents '(nil t nil t))))
    (declare (type (simple-array (member t nil) (4)) bswpiv xswpiv)
              (type (simple-array fixnum (4)) locu22 loc121 locu12))
    (defun dlasy2
      (ltranl ltranr isgn n1 n2 t1 ldt1 tr ldtr b ldb$ scale x ldx xnorm
       info)
      (declare (type (double-float) xnorm scale)
                (type (simple-array double-float (*)) x b tr t1)
                (type fixnum info ldx ldb$ ldtr ldt1 n2 n1 isgn)
                (type (member t nil) ltranr ltranl))
      (f2cl-lib:with-multi-array-data
        ((t1 double-float t1-%data% t1-%offset%)
         (tr double-float tr-%data% tr-%offset%)
         (b double-float b-%data% b-%offset%)
         (x double-float x-%data% x-%offset%))
        (prog ((btm (make-array 4 :element-type 'double-float))
               (t16 (make-array 16 :element-type 'double-float))
               (tmp (make-array 4 :element-type 'double-float))
               (x2 (make-array 2 :element-type 'double-float))
               (jpiv (make-array 4 :element-type 'fixnum)) (bet 0.0))
```



```

        (eps 0.0) (gam 0.0) (l21 0.0) (sgn 0.0) (smin 0.0) (smlnum 0.0)
        (tau1 0.0) (temp 0.0) (u11 0.0) (u12 0.0) (u22 0.0) (xmax 0.0)
        (i 0) (ip 0) (ipiv 0) (ipsv 0) (j 0) (jp 0) (jpsv 0) (k 0)
        (bswap nil) (xswap nil))
(declare (type (simple-array double-float (16)) t16)
         (type (simple-array double-float (4)) btmp tmp)
         (type (simple-array double-float (2)) x2)
         (type (simple-array fixnum (4)) jpiv)
         (type (double-float) bet eps gam l21 sgn smin smlnum tau1
               temp u11 u12 u22 xmax)
         (type fixnum i ip ipiv ipsv j jp jpsv k)
         (type (member t nil) bswap xswap))
(setf info 0)
(if (or (= n1 0) (= n2 0)) (go end_label))
(setf eps (dlamch "P"))
(setf smlnum (/ (dlamch "S") eps))
(setf sgn (coerce (the fixnum isgn) 'double-float))
(setf k (f2cl-lib:int-sub (f2cl-lib:int-add n1 n1 n2) 2))
(f2cl-lib:computed-goto (label10 label20 label30 label50) k)
label10
  (setf tau1
    (+
      (f2cl-lib:fref t1-%data% (1 1) ((1 ldt1) (1 *)) t1-%offset%)
      (* sgn
        (f2cl-lib:fref tr-%data%
          (1 1)
          ((1 ldtr) (1 *))
          tr-%offset%))))
  (setf bet (abs tau1))
  (cond
    ((<= bet smlnum)
     (setf tau1 smlnum)
     (setf bet smlnum)
     (setf info 1)))
  (setf scale one)
  (setf gam
    (abs
      (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%)))
  (if (> (* smlnum gam) bet) (setf scale (/ one gam)))
  (setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
    (/
      (*
        (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%)
        scale)
      tau1))
  (setf xnorm
    (abs
      (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)))
  (go end_label)
label20

```

```

(setf smin
  (max
    (* eps
      (max
        (abs
          (f2cl-lib:fref tl-%data%
                        (1 1)
                        ((1 ldtl) (1 *)))
          tl-%offset%))
        (abs
          (f2cl-lib:fref tr-%data%
                        (1 1)
                        ((1 ldtr) (1 *)))
          tr-%offset%))
        (abs
          (f2cl-lib:fref tr-%data%
                        (1 2)
                        ((1 ldtr) (1 *)))
          tr-%offset%))
        (abs
          (f2cl-lib:fref tr-%data%
                        (2 1)
                        ((1 ldtr) (1 *)))
          tr-%offset%))
        (abs
          (f2cl-lib:fref tr-%data%
                        (2 2)
                        ((1 ldtr) (1 *)))
          tr-%offset%))))
    smlnum))
(setf (f2cl-lib:fref tmp (1) ((1 4)))
  (+
    (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *))) tl-%offset%)
    (* sgn
      (f2cl-lib:fref tr-%data%
                    (1 1)
                    ((1 ldtr) (1 *)))
      tr-%offset%)))
(setf (f2cl-lib:fref tmp (4) ((1 4)))
  (+
    (f2cl-lib:fref tl-%data% (1 1) ((1 ldtl) (1 *))) tl-%offset%)
    (* sgn
      (f2cl-lib:fref tr-%data%
                    (2 2)
                    ((1 ldtr) (1 *)))
      tr-%offset%)))
(cond
  (ltranr
    (setf (f2cl-lib:fref tmp (2) ((1 4)))
      (* sgn
        (f2cl-lib:fref tr-%data%
                      (2 2)
                      ((1 ldtr) (1 *)))
        tr-%offset%)))

```

```

(f2cl-lib:fref tr-%data%
(2 1)
((1 ldtr) (1 *))
tr-%offset%)))
(setf (f2cl-lib:fref tmp (3) ((1 4)))
(* sgn
(f2cl-lib:fref tr-%data%
(1 2)
((1 ldtr) (1 *))
tr-%offset%)))
(t
(setf (f2cl-lib:fref tmp (2) ((1 4)))
(* sgn
(f2cl-lib:fref tr-%data%
(1 2)
((1 ldtr) (1 *))
tr-%offset%)))
(setf (f2cl-lib:fref tmp (3) ((1 4)))
(* sgn
(f2cl-lib:fref tr-%data%
(2 1)
((1 ldtr) (1 *))
tr-%offset%))))))
(setf (f2cl-lib:fref btmp (1) ((1 4)))
(f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))
(setf (f2cl-lib:fref btmp (2) ((1 4)))
(f2cl-lib:fref b-%data% (1 2) ((1 ldb$) (1 *)) b-%offset%))
(go label40)

label30
(setf smin
(max
(* eps
(max
(abs
(f2cl-lib:fref tr-%data%
(1 1)
((1 ldtr) (1 *))
tr-%offset%))
(abs
(f2cl-lib:fref tl-%data%
(1 1)
((1 ldtl) (1 *))
tl-%offset%))
(abs
(f2cl-lib:fref tl-%data%
(1 2)
((1 ldtl) (1 *))
tl-%offset%))
(abs
(f2cl-lib:fref tl-%data%
```

```

                                (2 1)
                                ((1 ldt1) (1 *))
                                tl-%offset%))
(abs
  (f2cl-lib:fref tl-%data%
    (2 2)
    ((1 ldt1) (1 *))
    tl-%offset%))))
  smlnum))
(setf (f2cl-lib:fref tmp (1) ((1 4)))
  (+
    (f2cl-lib:fref tl-%data% (1 1) ((1 ldt1) (1 *)) tl-%offset%)
    (* sgn
      (f2cl-lib:fref tr-%data%
        (1 1)
        ((1 ldtr) (1 *))
        tr-%offset%))))))
(setf (f2cl-lib:fref tmp (4) ((1 4)))
  (+
    (f2cl-lib:fref tl-%data% (2 2) ((1 ldt1) (1 *)) tl-%offset%)
    (* sgn
      (f2cl-lib:fref tr-%data%
        (1 1)
        ((1 ldtr) (1 *))
        tr-%offset%))))))
(cond
  (ltran1
    (setf (f2cl-lib:fref tmp (2) ((1 4)))
      (f2cl-lib:fref tl-%data%
        (1 2)
        ((1 ldt1) (1 *))
        tl-%offset%)))
    (setf (f2cl-lib:fref tmp (3) ((1 4)))
      (f2cl-lib:fref tl-%data%
        (2 1)
        ((1 ldt1) (1 *))
        tl-%offset%)))
  (t
    (setf (f2cl-lib:fref tmp (2) ((1 4)))
      (f2cl-lib:fref tl-%data%
        (2 1)
        ((1 ldt1) (1 *))
        tl-%offset%)))
    (setf (f2cl-lib:fref tmp (3) ((1 4)))
      (f2cl-lib:fref tl-%data%
        (1 2)
        ((1 ldt1) (1 *))
        tl-%offset%))))))
(setf (f2cl-lib:fref btmp (1) ((1 4)))
  (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))

```

```

      (setf (f2cl-lib:fref btmp (2) ((1 4)))
            (f2cl-lib:fref b-%data% (2 1) ((1 ldb$) (1 *)) b-%offset%))
label40
      (setf ipiv (idamax 4 tmp 1))
      (setf u11 (f2cl-lib:fref tmp (ipiv) ((1 4))))
      (cond
        ((<= (abs u11) smin)
         (setf info 1)
         (setf u11 smin)))
      (setf u12
            (f2cl-lib:fref tmp
                          ((f2cl-lib:fref locu12 (ipiv) ((1 4)))
                           ((1 4))))
            (/
             (f2cl-lib:fref tmp
                          ((f2cl-lib:fref loc121 (ipiv) ((1 4)))
                           ((1 4)))
             u11))
      (setf u22
            (-
             (f2cl-lib:fref tmp
                          ((f2cl-lib:fref locu22 (ipiv) ((1 4)))
                           ((1 4)))
             (* u12 121)))
      (setf xswap (f2cl-lib:fref xswpiv (ipiv) ((1 4))))
      (setf bswap (f2cl-lib:fref bswpiv (ipiv) ((1 4))))
      (cond
        ((<= (abs u22) smin)
         (setf info 1)
         (setf u22 smin)))
      (cond
        (bswap
         (setf temp (f2cl-lib:fref btmp (2) ((1 4))))
         (setf (f2cl-lib:fref btmp (2) ((1 4)))
               (- (f2cl-lib:fref btmp (1) ((1 4))) (* 121 temp)))
         (setf (f2cl-lib:fref btmp (1) ((1 4))) temp))
        (t
         (setf (f2cl-lib:fref btmp (2) ((1 4)))
               (- (f2cl-lib:fref btmp (2) ((1 4)))
                  (* 121 (f2cl-lib:fref btmp (1) ((1 4)))))))
      (setf scale one)
      (cond
        ((or
          (> (* two smlnum (abs (f2cl-lib:fref btmp (2) ((1 4))))
            (abs u22))
          (> (* two smlnum (abs (f2cl-lib:fref btmp (1) ((1 4))))
            (abs u11)))
         (setf scale
               (/ half

```

```

(max (abs (f2cl-lib:fref btmp (1) ((1 4))))
(abs (f2cl-lib:fref btmp (2) ((1 4))))))
(setf (f2cl-lib:fref btmp (1) ((1 4)))
(* (f2cl-lib:fref btmp (1) ((1 4))) scale))
(setf (f2cl-lib:fref btmp (2) ((1 4)))
(* (f2cl-lib:fref btmp (2) ((1 4))) scale))))
(setf (f2cl-lib:fref x2 (2) ((1 2)))
(/ (f2cl-lib:fref btmp (2) ((1 4))) u22))
(setf (f2cl-lib:fref x2 (1) ((1 2)))
(- (/ (f2cl-lib:fref btmp (1) ((1 4))) u11)
(* (/ u12 u11) (f2cl-lib:fref x2 (2) ((1 2))))))
(cond
(xswap
(setf temp (f2cl-lib:fref x2 (2) ((1 2))))
(setf (f2cl-lib:fref x2 (2) ((1 2)))
(f2cl-lib:fref x2 (1) ((1 2))))
(setf (f2cl-lib:fref x2 (1) ((1 2))) temp)))
(setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%)
(f2cl-lib:fref x2 (1) ((1 2))))
(cond
(= n1 1)
(setf (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%)
(f2cl-lib:fref x2 (2) ((1 2))))
(setf xnorm
(+
(abs
(f2cl-lib:fref x-%data%
(1 1)
((1 ldx) (1 *))
x-%offset%))
(abs
(f2cl-lib:fref x-%data%
(1 2)
((1 ldx) (1 *))
x-%offset%))))))
(t
(setf (f2cl-lib:fref x-%data% (2 1) ((1 ldx) (1 *)) x-%offset%)
(f2cl-lib:fref x2 (2) ((1 2))))
(setf xnorm
(max
(abs
(f2cl-lib:fref x-%data%
(1 1)
((1 ldx) (1 *))
x-%offset%))
(abs
(f2cl-lib:fref x-%data%
(2 1)
((1 ldx) (1 *))
x-%offset%))))))

```

```

      (go end_label)
label50
      (setf smin
        (max
          (abs
            (f2cl-lib:fref tr-%data%
                          (1 1)
                          ((1 ldtr) (1 *))
                          tr-%offset%))
          (abs
            (f2cl-lib:fref tr-%data%
                          (1 2)
                          ((1 ldtr) (1 *))
                          tr-%offset%))
          (abs
            (f2cl-lib:fref tr-%data%
                          (2 1)
                          ((1 ldtr) (1 *))
                          tr-%offset%))
          (abs
            (f2cl-lib:fref tr-%data%
                          (2 2)
                          ((1 ldtr) (1 *))
                          tr-%offset%))))
      (setf smin
        (max smin
          (abs
            (f2cl-lib:fref t1-%data%
                          (1 1)
                          ((1 ldt1) (1 *))
                          t1-%offset%))
          (abs
            (f2cl-lib:fref t1-%data%
                          (1 2)
                          ((1 ldt1) (1 *))
                          t1-%offset%))
          (abs
            (f2cl-lib:fref t1-%data%
                          (2 1)
                          ((1 ldt1) (1 *))
                          t1-%offset%))
          (abs
            (f2cl-lib:fref t1-%data%
                          (2 2)
                          ((1 ldt1) (1 *))
                          t1-%offset%))))
      (setf smin (max (* eps smin) smlnum))
      (setf (f2cl-lib:fref btmp (1) ((1 4))) zero)
      (dcopy 16 btmp 0 t16 1)
      (setf (f2cl-lib:fref t16 (1 1) ((1 4) (1 4)))

```

```

(
  (+
    (f2cl-lib:fref tl-%data% (1 1) ((1 ldt1) (1 *)) tl-%offset%)
    (* sgn
      (f2cl-lib:fref tr-%data%
        (1 1)
        ((1 ldtr) (1 *))
        tr-%offset%))))
  (setf (f2cl-lib:fref t16 (2 2) ((1 4) (1 4)))
    (+
      (f2cl-lib:fref tl-%data% (2 2) ((1 ldt1) (1 *)) tl-%offset%)
      (* sgn
        (f2cl-lib:fref tr-%data%
          (1 1)
          ((1 ldtr) (1 *))
          tr-%offset%))))
    (setf (f2cl-lib:fref t16 (3 3) ((1 4) (1 4)))
      (+
        (f2cl-lib:fref tl-%data% (1 1) ((1 ldt1) (1 *)) tl-%offset%)
        (* sgn
          (f2cl-lib:fref tr-%data%
            (2 2)
            ((1 ldtr) (1 *))
            tr-%offset%))))
        (setf (f2cl-lib:fref t16 (4 4) ((1 4) (1 4)))
          (+
            (f2cl-lib:fref tl-%data% (2 2) ((1 ldt1) (1 *)) tl-%offset%)
            (* sgn
              (f2cl-lib:fref tr-%data%
                (2 2)
                ((1 ldtr) (1 *))
                tr-%offset%))))
            (cond
              (ltranl
                (setf (f2cl-lib:fref t16 (1 2) ((1 4) (1 4)))
                  (f2cl-lib:fref tl-%data%
                    (2 1)
                    ((1 ldt1) (1 *))
                    tl-%offset%))
                (setf (f2cl-lib:fref t16 (2 1) ((1 4) (1 4)))
                  (f2cl-lib:fref tl-%data%
                    (1 2)
                    ((1 ldt1) (1 *))
                    tl-%offset%))
                (setf (f2cl-lib:fref t16 (3 4) ((1 4) (1 4)))
                  (f2cl-lib:fref tl-%data%
                    (2 1)
                    ((1 ldt1) (1 *))
                    tl-%offset%))
                (setf (f2cl-lib:fref t16 (4 3) ((1 4) (1 4)))
                  (f2cl-lib:fref tl-%data%

```



```

(1 2)
((1 ldt1) (1 *))
tl-%offset%)))
(t
  (setf (f2cl-lib:fref t16 (1 2) ((1 4) (1 4)))
    (f2cl-lib:fref tl-%data%
      (1 2)
      ((1 ldt1) (1 *))
      tl-%offset%))
  (setf (f2cl-lib:fref t16 (2 1) ((1 4) (1 4)))
    (f2cl-lib:fref tl-%data%
      (2 1)
      ((1 ldt1) (1 *))
      tl-%offset%))
  (setf (f2cl-lib:fref t16 (3 4) ((1 4) (1 4)))
    (f2cl-lib:fref tl-%data%
      (1 2)
      ((1 ldt1) (1 *))
      tl-%offset%))
  (setf (f2cl-lib:fref t16 (4 3) ((1 4) (1 4)))
    (f2cl-lib:fref tl-%data%
      (2 1)
      ((1 ldt1) (1 *))
      tl-%offset%))))
(cond
  (ltranr
    (setf (f2cl-lib:fref t16 (1 3) ((1 4) (1 4)))
      (* sgn
        (f2cl-lib:fref tr-%data%
          (1 2)
          ((1 ldtr) (1 *))
          tr-%offset%)))
    (setf (f2cl-lib:fref t16 (2 4) ((1 4) (1 4)))
      (* sgn
        (f2cl-lib:fref tr-%data%
          (1 2)
          ((1 ldtr) (1 *))
          tr-%offset%)))
    (setf (f2cl-lib:fref t16 (3 1) ((1 4) (1 4)))
      (* sgn
        (f2cl-lib:fref tr-%data%
          (2 1)
          ((1 ldtr) (1 *))
          tr-%offset%)))
    (setf (f2cl-lib:fref t16 (4 2) ((1 4) (1 4)))
      (* sgn
        (f2cl-lib:fref tr-%data%
          (2 1)
          ((1 ldtr) (1 *))
          tr-%offset%))))

```

```

(t
  (setf (f2cl-lib:fref t16 (1 3) ((1 4) (1 4)))
    (* sgn
      (f2cl-lib:fref tr-%data%
        (2 1)
        ((1 ldtr) (1 *))
        tr-%offset%)))
  (setf (f2cl-lib:fref t16 (2 4) ((1 4) (1 4)))
    (* sgn
      (f2cl-lib:fref tr-%data%
        (2 1)
        ((1 ldtr) (1 *))
        tr-%offset%)))
  (setf (f2cl-lib:fref t16 (3 1) ((1 4) (1 4)))
    (* sgn
      (f2cl-lib:fref tr-%data%
        (1 2)
        ((1 ldtr) (1 *))
        tr-%offset%)))
  (setf (f2cl-lib:fref t16 (4 2) ((1 4) (1 4)))
    (* sgn
      (f2cl-lib:fref tr-%data%
        (1 2)
        ((1 ldtr) (1 *))
        tr-%offset%))))
  (setf (f2cl-lib:fref btmp (1) ((1 4)))
    (f2cl-lib:fref b-%data% (1 1) ((1 ldb$) (1 *)) b-%offset%))
  (setf (f2cl-lib:fref btmp (2) ((1 4)))
    (f2cl-lib:fref b-%data% (2 1) ((1 ldb$) (1 *)) b-%offset%))
  (setf (f2cl-lib:fref btmp (3) ((1 4)))
    (f2cl-lib:fref b-%data% (1 2) ((1 ldb$) (1 *)) b-%offset%))
  (setf (f2cl-lib:fref btmp (4) ((1 4)))
    (f2cl-lib:fref b-%data% (2 2) ((1 ldb$) (1 *)) b-%offset%))
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i 3) nil)
  (tagbody
    (setf xmax zero)
    (f2cl-lib:fdo (ip i (f2cl-lib:int-add ip 1))
      (> ip 4) nil)
    (tagbody
      (f2cl-lib:fdo (jp i (f2cl-lib:int-add jp 1))
        (> jp 4) nil)
      (tagbody
        (cond
          ((>= (abs (f2cl-lib:fref t16 (ip jp) ((1 4) (1 4))))
            xmax)
            (setf xmax
              (abs
                (f2cl-lib:fref t16 (ip jp) ((1 4) (1 4))))))
          (setf ipsv ip)

```

```

                (setf jpsv jp))))))
(cond
  ((/= ipsv i)
    (dswap 4
      (f2cl-lib:array-slice t16
        double-float
        (ipsv 1)
        ((1 4) (1 4)))
      4 (f2cl-lib:array-slice t16 double-float (i 1) ((1 4) (1 4)))
      4)
    (setf temp (f2cl-lib:fref btmp (i) ((1 4))))
    (setf (f2cl-lib:fref btmp (i) ((1 4)))
      (f2cl-lib:fref btmp (ipsv) ((1 4))))
    (setf (f2cl-lib:fref btmp (ipsv) ((1 4))) temp)))
  (if (/= jpsv i)
    (dswap 4
      (f2cl-lib:array-slice t16
        double-float
        (1 jpsv)
        ((1 4) (1 4)))
      1
      (f2cl-lib:array-slice t16 double-float (1 i) ((1 4) (1 4)))
      1))
    (setf (f2cl-lib:fref jpiv (i) ((1 4))) jpsv)
    (cond
      ((< (abs (f2cl-lib:fref t16 (i i) ((1 4) (1 4)))) smin)
        (setf info 1)
        (setf (f2cl-lib:fref t16 (i i) ((1 4) (1 4))) smin)))
      (f2cl-lib:fdo (j (f2cl-lib:int-add i 1) (f2cl-lib:int-add j 1))
        ((> j 4) nil)
        (tagbody
          (setf (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
            (/ (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
              (f2cl-lib:fref t16 (i i) ((1 4) (1 4)))))
          (setf (f2cl-lib:fref btmp (j) ((1 4)))
            (- (f2cl-lib:fref btmp (j) ((1 4)))
              (* (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                 (f2cl-lib:fref btmp (i) ((1 4)))))))
          (f2cl-lib:fdo (k (f2cl-lib:int-add i 1)
            (f2cl-lib:int-add k 1))
            ((> k 4) nil)
            (tagbody
              (setf (f2cl-lib:fref t16 (j k) ((1 4) (1 4)))
                (- (f2cl-lib:fref t16 (j k) ((1 4) (1 4)))
                  (* (f2cl-lib:fref t16 (j i) ((1 4) (1 4)))
                     (f2cl-lib:fref t16 (i k) ((1 4) (1 4))))))))))
          (if (< (abs (f2cl-lib:fref t16 (4 4) ((1 4) (1 4)))) smin)
            (setf (f2cl-lib:fref t16 (4 4) ((1 4) (1 4))) smin))
          (setf scale one)
          (cond

```

```

(or
  (> (* eight smlnum (abs (f2cl-lib:fref btmp (1) ((1 4))))
    (abs (f2cl-lib:fref t16 (1 1) ((1 4) (1 4)))))
  (> (* eight smlnum (abs (f2cl-lib:fref btmp (2) ((1 4))))
    (abs (f2cl-lib:fref t16 (2 2) ((1 4) (1 4)))))
  (> (* eight smlnum (abs (f2cl-lib:fref btmp (3) ((1 4))))
    (abs (f2cl-lib:fref t16 (3 3) ((1 4) (1 4)))))
  (> (* eight smlnum (abs (f2cl-lib:fref btmp (4) ((1 4))))
    (abs (f2cl-lib:fref t16 (4 4) ((1 4) (1 4))))))
(setf scale
  (/ (/ one eight)
    (max (abs (f2cl-lib:fref btmp (1) ((1 4)))
      (abs (f2cl-lib:fref btmp (2) ((1 4)))
      (abs (f2cl-lib:fref btmp (3) ((1 4)))
      (abs (f2cl-lib:fref btmp (4) ((1 4))))))
(setf (f2cl-lib:fref btmp (1) ((1 4)))
  (* (f2cl-lib:fref btmp (1) ((1 4))) scale))
(setf (f2cl-lib:fref btmp (2) ((1 4)))
  (* (f2cl-lib:fref btmp (2) ((1 4))) scale))
(setf (f2cl-lib:fref btmp (3) ((1 4)))
  (* (f2cl-lib:fref btmp (3) ((1 4))) scale))
(setf (f2cl-lib:fref btmp (4) ((1 4)))
  (* (f2cl-lib:fref btmp (4) ((1 4))) scale))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i 4) nil)
(tagbody
  (setf k (f2cl-lib:int-sub 5 i))
  (setf temp (/ one (f2cl-lib:fref t16 (k k) ((1 4) (1 4)))))
  (setf (f2cl-lib:fref tmp (k) ((1 4)))
    (* (f2cl-lib:fref btmp (k) ((1 4))) temp))
  (f2cl-lib:fdo (j (f2cl-lib:int-add k 1) (f2cl-lib:int-add j 1))
    ((> j 4) nil)
    (tagbody
      (setf (f2cl-lib:fref tmp (k) ((1 4)))
        (- (f2cl-lib:fref tmp (k) ((1 4)))
          (* temp
            (f2cl-lib:fref t16 (k j) ((1 4) (1 4)))
            (f2cl-lib:fref tmp (j) ((1 4))))))))))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i 3) nil)
(tagbody
  (cond
    ((/=
      (f2cl-lib:fref jpiv
        ((f2cl-lib:int-add 4 (f2cl-lib:int-sub i)))
        ((1 4)))
      (f2cl-lib:int-add 4 (f2cl-lib:int-sub i)))
    (setf temp
      (f2cl-lib:fref tmp ((f2cl-lib:int-sub 4 i)) ((1 4))))
    (setf (f2cl-lib:fref tmp ((f2cl-lib:int-sub 4 i)) ((1 4)))

```

```

(f2cl-lib:fref tmp
  ((f2cl-lib:fref jpiv
    ((f2cl-lib:int-sub 4
      i))
    ((1 4))))
  ((1 4)))
(setf (f2cl-lib:fref tmp
  ((f2cl-lib:fref jpiv
    ((f2cl-lib:int-sub 4 i))
    ((1 4))))
  temp))))
(setf (f2cl-lib:fref x-%data% (1 1) ((1 ldx) (1 *)) x-%offset%
  (f2cl-lib:fref tmp (1) ((1 4))))
  (f2cl-lib:fref tmp (2) ((1 4))))
(setf (f2cl-lib:fref x-%data% (2 1) ((1 ldx) (1 *)) x-%offset%
  (f2cl-lib:fref tmp (2) ((1 4))))
  (f2cl-lib:fref tmp (3) ((1 4))))
(setf (f2cl-lib:fref x-%data% (1 2) ((1 ldx) (1 *)) x-%offset%
  (f2cl-lib:fref tmp (3) ((1 4))))
  (f2cl-lib:fref tmp (4) ((1 4))))
(setf xnorm
  (max
    (+ (abs (f2cl-lib:fref tmp (1) ((1 4))))
      (abs (f2cl-lib:fref tmp (3) ((1 4))))
      (+ (abs (f2cl-lib:fref tmp (2) ((1 4))))
        (abs (f2cl-lib:fref tmp (4) ((1 4)))))))
    end_label
  (return
    (values nil
      nil
      nil
      nil
      nil
      nil
      nil
      nil
      nil
      nil
      nil
      scale
      nil
      nil
      xnorm
      info))))))

```

dorg2r LAPACK**— dorg2r.input —**

```

)set break resume
)sys rm -f dorg2r.output
)spool dorg2r.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorg2r.help —

```

=====
dorg2r examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DORG2R - an m by n real matrix Q with orthonormal columns,

SYNOPSIS

```
SUBROUTINE DORG2R( M, N, K, A, LDA, TAU, WORK, INFO )
```

```
      INTEGER      INFO, K, LDA, M, N
```

```
      DOUBLE      PRECISION A( LDA, * ), TAU( * ), WORK( * )
```

Purpose

```
=====
```

DORG2R generates an m by n real matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors of order m

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF.

Arguments

=====

M (input) INTEGER
 The number of rows of the matrix Q. $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix Q. $M \geq N \geq 0$.

K (input) INTEGER
 The number of elementary reflectors whose product defines the
 matrix Q. $N \geq K \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
 On entry, the i-th column must contain the vector which
 defines the elementary reflector $H(i)$, for $i = 1, 2, \dots, k$, as
 returned by DGEQRF in the first k columns of its array
 argument A.
 On exit, the m-by-n matrix Q.

LDA (input) INTEGER
 The first dimension of the array A. $LDA \geq \max(1, M)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
 TAU(i) must contain the scalar factor of the elementary
 reflector $H(i)$, as returned by DGEQRF.

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument has an illegal value

— dorg2r.f —

```

SUBROUTINE DORG2R( M, N, K, A, LDA, TAU, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
*  INTEGER          INFO, K, LDA, M, N
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )

```

```

*      ..
*
*      =====
*
*      .. Parameters ..
      DOUBLE PRECISION    ONE, ZERO
      PARAMETER            ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      ..
*      .. Local Scalars ..
      INTEGER              I, J, L
*
*      ..
*      .. External Subroutines ..
      EXTERNAL             DLARF, DSCAL, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC            MAX
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 .OR. N.GT.M ) THEN
         INFO = -2
      ELSE IF( K.LT.0 .OR. K.GT.N ) THEN
         INFO = -3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DORG2R', -INFO )
         RETURN
      END IF
*
*      Quick return if possible
*
      IF( N.LE.0 )
$    RETURN
*
*      Initialise columns k+1:n to columns of the unit matrix
*
      DO 20 J = K + 1, N
         DO 10 L = 1, M
            A( L, J ) = ZERO
10        CONTINUE
         A( J, J ) = ONE
20    CONTINUE
*

```



```

      DO 40 I = K, 1, -1
*
*      Apply H(i) to A(i:m,i:n) from the left
*
      IF( I.LT.N ) THEN
        A( I, I ) = ONE
        CALL DLARF( 'Left', M-I+1, N-I, A( I, I ), 1, TAU( I ),
$          A( I, I+1 ), LDA, WORK )
      END IF
      IF( I.LT.M )
$        CALL DSCAL( M-I, -TAU( I ), A( I+1, I ), 1 )
      A( I, I ) = ONE - TAU( I )
*
*      Set A(1:i-1,i) to zero
*
      DO 30 L = 1, I - 1
        A( L, I ) = ZERO
30    CONTINUE
40  CONTINUE
      RETURN
*
*      End of DORG2R
*
      END

```

— LAPACK dorg2r —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dorg2r (m n k a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lda k n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (j 0) (l 0))
        (declare (type fixnum i j l))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info -1))
          ((or (< n 0) (> n m))
            (setf info -2))
          ((or (< k 0) (> k n))

```

```

(setf info -3))
(< lda (max (the fixnum 1) (the fixnum m)))
(setf info -5)))
(cond
  (/= info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DORG2R" (f2cl-lib:int-sub info))
  (go end_label)))
(if (<= n 0) (go end_label))
(f2cl-lib:fdo (j (f2cl-lib:int-add k 1) (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
    (> l m) nil)
  (tagbody
    (setf (f2cl-lib:fref a-%data% (l j) ((1 lda) (1 *)) a-%offset%
      zero)))
    (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%
      one)))
  (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    (> i 1) nil)
  (tagbody
    (cond
      (< i n)
      (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%
        one)
      (dlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
        (f2cl-lib:int-sub n i)
        (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
        (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
        (f2cl-lib:array-slice a
          double-float
          (i (f2cl-lib:int-add i 1))
          ((1 lda) (1 *)))
        lda work)))
      (if (< i m)
        (dscal (f2cl-lib:int-sub m i)
          (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
            (f2cl-lib:array-slice a
              double-float
              ((+ i 1) i)
              ((1 lda) (1 *)))
            1))
        (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
          (- one
            (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
          (> l (f2cl-lib:int-add i (f2cl-lib:int-sub 1))) nil)
        (tagbody

```

```

                (setf (f2cl-lib:fref a-%data% (1 i) ((1 lda) (1 *)) a-%offset%)
                    zero))))
end_label
    (return (values nil nil nil nil nil nil nil info))))))

```

dorgbr LAPACK

— dorgbr.input —

```

)set break resume
)sys rm -f dorgbr.output
)spool dorgbr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorgbr.help —

```

=====
dorgbr examples
=====

=====
Man Page Details
=====

```

NAME

DORGBR - one of the real orthogonal matrices Q or P**T determined by DGEBRD when reducing a real matrix A to bidiagonal form

SYNOPSIS

SUBROUTINE DORGBR(VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO)

| | |
|-----------|--|
| CHARACTER | VECT |
| INTEGER | INFO, K, LDA, LWORK, M, N |
| DOUBLE | PRECISION A(LDA, *), TAU(*), WORK(*) |

Purpose

=====

DORGBR generates one of the real orthogonal matrices Q or P**T determined by DGEBRD when reducing a real matrix A to bidiagonal form: $A = Q * B * P^{**T}$. Q and P**T are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = 'Q', A is assumed to have been an M-by-K matrix, and Q is of order M:
 if $m \geq k$, $Q = H(1) H(2) \dots H(k)$ and DORGBR returns the first n columns of Q, where $m \geq n \geq k$;
 if $m < k$, $Q = H(1) H(2) \dots H(m-1)$ and DORGBR returns Q as an M-by-M matrix.

If VECT = 'P', A is assumed to have been a K-by-N matrix, and P**T is of order N:
 if $k < n$, $P^{**T} = G(k) \dots G(2) G(1)$ and DORGBR returns the first m rows of P**T, where $n \geq m \geq k$;
 if $k \geq n$, $P^{**T} = G(n-1) \dots G(2) G(1)$ and DORGBR returns P**T as an N-by-N matrix.

Arguments

=====

- VECT (input) CHARACTER*1
 Specifies whether the matrix Q or the matrix P**T is required, as defined in the transformation applied by DGEBRD:
 = 'Q': generate Q;
 = 'P': generate P**T.
- M (input) INTEGER
 The number of rows of the matrix Q or P**T to be returned.
 $M \geq 0$.
- N (input) INTEGER
 The number of columns of the matrix Q or P**T to be returned.
 $N \geq 0$.
 If VECT = 'Q', $M \geq N \geq \min(M, K)$;
 if VECT = 'P', $N \geq M \geq \min(N, K)$.
- K (input) INTEGER
 If VECT = 'Q', the number of columns in the original M-by-K matrix reduced by DGEBRD.
 If VECT = 'P', the number of rows in the original K-by-N matrix reduced by DGEBRD.
 $K \geq 0$.
- A (input/output) DOUBLE PRECISION array, dimension (LDA,N)

On entry, the vectors which define the elementary reflectors,
as returned by DGEBRD.

On exit, the M-by-N matrix Q or P**T.

LDA (input) INTEGER
The leading dimension of the array A. LDA \geq max(1,M).

TAU (input) DOUBLE PRECISION array, dimension
(min(M,K)) if VECT = 'Q'
(min(N,K)) if VECT = 'P'
TAU(i) must contain the scalar factor of the elementary
reflector H(i) or G(i), which determines Q or P**T, as
returned by DGEBRD in its array argument TAUQ or TAUP.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK \geq max(1,min(M,N)).
For optimum performance LWORK \geq min(M,N)*NB, where NB
is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

— dorgbr.f —

```

SUBROUTINE DORGBR( VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
CHARACTER          VECT
INTEGER            INFO, K, LDA, LWORK, M, N
*
*  .. Array Arguments ..
DOUBLE PRECISION   A( LDA, * ), TAU( * ), WORK( * )

```

```

*      ..
*
*      =====
*
*      .. Parameters ..
      DOUBLE PRECISION    ZERO, ONE
      PARAMETER            ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*      ..
*      .. Local Scalars ..
      LOGICAL              LQUERY, WANTQ
      INTEGER              I, IINFO, J, LWKOPT, MN, NB
*
*      ..
*      .. External Functions ..
      LOGICAL              LSAME
      INTEGER              ILAENV
      EXTERNAL             LSAME, ILAENV
*
*      ..
*      .. External Subroutines ..
      EXTERNAL             DORGLQ, DORGQR, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC            MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      WANTQ = LSAME( VECT, 'Q' )
      MN = MIN( M, N )
      LQUERY = ( LWORK.EQ.-1 )
      IF( .NOT.WANTQ .AND. .NOT.LSAME( VECT, 'P' ) ) THEN
         INFO = -1
      ELSE IF( M.LT.0 ) THEN
         INFO = -2
      ELSE IF( N.LT.0 .OR. ( WANTQ .AND. ( N.GT.M .OR. N.LT.MIN( M,
$           K ) ) ) .OR. ( .NOT.WANTQ .AND. ( M.GT.N .OR. M.LT.
$           MIN( N, K ) ) ) ) THEN
         INFO = -3
      ELSE IF( K.LT.0 ) THEN
         INFO = -4
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -6
      ELSE IF( LWORK.LT.MAX( 1, MN ) .AND. .NOT.LQUERY ) THEN
         INFO = -9
      END IF
*
      IF( INFO.EQ.0 ) THEN
         IF( WANTQ ) THEN
            NB = ILAENV( 1, 'DORGQR', ' ', M, N, K, -1 )

```

```

        ELSE
            NB = ILAENV( 1, 'DORGLQ', ' ', M, N, K, -1 )
        END IF
        LWKOPT = MAX( 1, MN ) * NB
        WORK( 1 ) = LWKOPT
    END IF
*
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORGBR', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Quick return if possible
*
    IF( M.EQ.0 .OR. N.EQ.0 ) THEN
        WORK( 1 ) = 1
        RETURN
    END IF
*
    IF( WANTQ ) THEN
*
*       Form Q, determined by a call to DGEHRD to reduce an m-by-k
*       matrix
*
        IF( M.GE.K ) THEN
*
*           If m >= k, assume m >= n >= k
*
            CALL DORGQR( M, N, K, A, LDA, TAU, WORK, LWORK, IINFO )
*
        ELSE
*
*           If m < k, assume m = n
*
            Shift the vectors which define the elementary reflectors one
            column to the right, and set the first row and column of Q
            to those of the unit matrix
*
            DO 20 J = M, 2, -1
                A( 1, J ) = ZERO
                DO 10 I = J + 1, M
                    A( I, J ) = A( I, J-1 )
10             CONTINUE
20             CONTINUE
            A( 1, 1 ) = ONE
            DO 30 I = 2, M
                A( I, 1 ) = ZERO
30             CONTINUE

```

```

      IF( M.GT.1 ) THEN
*
*      Form Q(2:m,2:m)
*
      CALL DORGQR( M-1, M-1, M-1, A( 2, 2 ), LDA, TAU, WORK,
$          LWORK, IINFO )
      END IF
      END IF
      ELSE
*
*      Form P', determined by a call to DGEHRD to reduce a k-by-n
*      matrix
*
      IF( K.LT.N ) THEN
*
*      If k < n, assume k <= m <= n
*
      CALL DORGLQ( M, N, K, A, LDA, TAU, WORK, LWORK, IINFO )
*
      ELSE
*
*      If k >= n, assume m = n
*
*      Shift the vectors which define the elementary reflectors one
*      row downward, and set the first row and column of P' to
*      those of the unit matrix
*
      A( 1, 1 ) = ONE
      DO 40 I = 2, N
        A( I, 1 ) = ZERO
40      CONTINUE
      DO 60 J = 2, N
        DO 50 I = J - 1, 2, -1
          A( I, J ) = A( I-1, J )
50      CONTINUE
        A( 1, J ) = ZERO
60      CONTINUE
      IF( N.GT.1 ) THEN
*
*      Form P'(2:n,2:n)
*
      CALL DORGLQ( N-1, N-1, N-1, A( 2, 2 ), LDA, TAU, WORK,
$          LWORK, IINFO )
      END IF
      END IF
      WORK( 1 ) = LWKOPT
      RETURN
*
*      End of DORGBR

```


*

END

— LAPACK dorgbr —

```
(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dorgbr (vect m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lwork lda k n m)
              (type character vect))
    (f2cl-lib:with-multi-array-data
      ((vect character vect-%data% vect-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (lwkopt 0) (mn 0) (nb 0) (lquery nil)
              (wantq nil))
        (declare (type fixnum i iinfo j lwkopt mn nb)
                  (type (member t nil) lquery wantq))
        (setf info 0)
        (setf wantq (char-equal vect #\Q))
        (setf mn (min (the fixnum m) (the fixnum n)))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((and (not wantq) (not (char-equal vect #\P)))
           (setf info -1))
          ((< m 0)
           (setf info -2))
          ((or (< n 0)
               (and wantq
                    (or (> n m)
                        (< n
                          (min (the fixnum m)
                                (the fixnum k))))))
           (and (not wantq)
                (or (> m n)
                    (< m
                     (min (the fixnum n)
                           (the fixnum k))))))
          (setf info -3))
          ((< k 0)
           (setf info -4))
          ((< lda (max (the fixnum 1) (the fixnum m)))
           (setf info -6))
```

```

((and
  (< lwork
    (max (the fixnum 1) (the fixnum mn)))
  (not lquery))
  (setf info -9)))
(cond
  ((= info 0)
    (cond
      (wantq
        (setf nb (ilaenv 1 "DORGQR" " " m n k -1)))
      (t
        (setf nb (ilaenv 1 "DORGLQ" " " m n k -1))))
    (setf lwkopt
      (f2cl-lib:int-mul
        (max (the fixnum 1) (the fixnum mn))
        nb))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum lwkopt) 'double-float))))
p (cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DORGBR" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(cond
  ((or (= m 0) (= n 0))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum 1) 'double-float))
    (go end_label)))
(cond
  (wantq
    (cond
      ((>= m k)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
          (dorgqr m n k a lda tau work lwork iinfo)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7))
          (setf iinfo var-8)))
      (t
        (f2cl-lib:fdo (j m (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
          ((> j 2) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data%
              (1 j)
              ((1 lda) (1 *))
              a-%offset%)
              zero)

```

```

(f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                (f2cl-lib:int-add i 1))
              (> i m) nil)
(tagbody
 (setf (f2cl-lib:fref a-%data%
                     (i j)
                     ((1 lda) (1 *))
                     a-%offset%)
       (f2cl-lib:fref a-%data%
                     (i (f2cl-lib:int-sub j 1))
                     ((1 lda) (1 *))
                     a-%offset%))))))
(setf (f2cl-lib:fref a-%data% (1 1) ((1 lda) (1 *)) a-%offset%)
      one)
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              (> i m) nil)
(tagbody
 (setf (f2cl-lib:fref a-%data%
                     (i 1)
                     ((1 lda) (1 *))
                     a-%offset%)
       zero)))
(cond
 (> m 1)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (dorgqr (f2cl-lib:int-sub m 1) (f2cl-lib:int-sub m 1)
          (f2cl-lib:int-sub m 1)
          (f2cl-lib:array-slice a
                                double-float
                                (2 2)
                                ((1 lda) (1 *)))
          lda tau work lwork iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7))
  (setf iinfo var-8))))))
(t
 (cond
 (< k n)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (dorglq m n k a lda tau work lwork iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                    var-7))
  (setf iinfo var-8)))
 (t
  (setf (f2cl-lib:fref a-%data% (1 1) ((1 lda) (1 *)) a-%offset%)
        one)
  (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                (> i n) nil)

```

```

(tagbody
  (setf (f2cl-lib:fref a-%data%
                     (i 1)
                     ((1 lda) (1 *))
                     a-%offset%)
        zero)))
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
  ((> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))
                  (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
      ((> i 2) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
                           (i j)
                           ((1 lda) (1 *))
                           a-%offset%)
              (f2cl-lib:fref a-%data%
                           ((f2cl-lib:int-sub i 1) j)
                           ((1 lda) (1 *))
                           a-%offset%))))
        (setf (f2cl-lib:fref a-%data%
                           (1 j)
                           ((1 lda) (1 *))
                           a-%offset%)
              zero)))
(cond
  ((> n 1)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (dorglq (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1)
             (f2cl-lib:int-sub n 1)
             (f2cl-lib:array-slice a
                                   double-float
                                   (2 2)
                                   ((1 lda) (1 *)))
             lda tau work lwork iinfo)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                      var-7))
     (setf iinfo var-8))))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum lwkopt) 'double-float))
end_label
(return (values nil nil nil nil nil nil nil nil info))))

```

dorghr LAPACK**— dorghr.input —**

```

)set break resume
)sys rm -f dorghr.output
)spool dorghr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorghr.help —

```

=====
dorghr examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DORGHR - a real orthogonal matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by DGEHRD

SYNOPSIS

```
SUBROUTINE DORGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
```

```

      INTEGER          IHI, ILO, INFO, LDA, LWORK, N

      DOUBLE           PRECISION A( LDA, * ), TAU( * ), WORK( * )

```

Purpose

```
=====
```

DORGHR generates a real orthogonal matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by DGEHRD:

$$Q = H(i_{lo}) H(i_{lo}+1) \dots H(i_{hi}-1).$$
Arguments

```
=====
```

N (input) INTEGER
The order of the matrix Q. $N \geq 0$.

ILO (input) INTEGER
IHI (input) INTEGER
ILO and IHI must have the same values as in the previous call of DGEHRD. Q is equal to the unit matrix except in the submatrix $Q(i_{lo}+1:i_{hi}, i_{lo}+1:i_{hi})$.
 $1 \leq ILO \leq IHI \leq N$, if $N > 0$; $ILO=1$ and $IHI=0$, if $N=0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the vectors which define the elementary reflectors, as returned by DGEHRD.
On exit, the N-by-N orthogonal matrix Q.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1, N)$.

TAU (input) DOUBLE PRECISION array, dimension (N-1)
 $TAU(i)$ must contain the scalar factor of the elementary reflector $H(i)$, as returned by DGEHRD.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if $INFO = 0$, $WORK(1)$ returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq IHI - ILO$.
For optimum performance $LWORK \geq (IHI - ILO) * NB$, where NB is the optimal blocksize.

If $LWORK = -1$, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if $INFO = -i$, the i-th argument had an illegal value

— dorghr.f —

```

SUBROUTINE DORGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
*
* -- LAPACK routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
```

```
* Courant Institute, Argonne National Lab, and Rice University
* June 30, 1999
*
```

```
* .. Scalar Arguments ..
INTEGER      IHI, ILO, INFO, LDA, LWORK, N
*
* ..
* .. Array Arguments ..
DOUBLE PRECISION  A( LDA, * ), TAU( * ), WORK( * )
*
* ..
```

```
* =====
```

```
* .. Parameters ..
DOUBLE PRECISION  ZERO, ONE
PARAMETER         ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
* ..
```

```
* .. Local Scalars ..
LOGICAL          LQUERY
INTEGER          I, IINFO, J, LWKOPT, NB, NH
*
* ..
```

```
* .. External Subroutines ..
EXTERNAL         DORGQR, XERBLA
*
* ..
```

```
* .. External Functions ..
INTEGER          ILAENV
EXTERNAL         ILAENV
*
* ..
```

```
* .. Intrinsic Functions ..
INTRINSIC        MAX, MIN
*
* ..
```

```
* .. Executable Statements ..
```

```
* Test the input arguments
```

```
*
INFO = 0
NH = IHI - ILO
LQUERY = ( LWORK.EQ.-1 )
IF( N.LT.0 ) THEN
    INFO = -1
ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
    INFO = -2
ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
    INFO = -3
ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
    INFO = -5
ELSE IF( LWORK.LT.MAX( 1, NH ) .AND. .NOT.LQUERY ) THEN
    INFO = -8
END IF
*
IF( INFO.EQ.0 ) THEN
```

```

        NB = ILAENV( 1, 'DORGQR', ' ', NH, NH, NH, -1 )
        LWKOPT = MAX( 1, NH )*NB
        WORK( 1 ) = LWKOPT
    END IF
*
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORGHR', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Quick return if possible
*
    IF( N.EQ.0 ) THEN
        WORK( 1 ) = 1
        RETURN
    END IF
*
*   Shift the vectors which define the elementary reflectors one
*   column to the right, and set the first ilo and the last n-ihl
*   rows and columns to those of the unit matrix
*
    DO 40 J = IHI, ILO + 1, -1
        DO 10 I = 1, J - 1
            A( I, J ) = ZERO
10    CONTINUE
        DO 20 I = J + 1, IHI
            A( I, J ) = A( I, J-1 )
20    CONTINUE
        DO 30 I = IHI + 1, N
            A( I, J ) = ZERO
30    CONTINUE
40 CONTINUE
    DO 60 J = 1, ILO
        DO 50 I = 1, N
            A( I, J ) = ZERO
50    CONTINUE
        A( J, J ) = ONE
60 CONTINUE
    DO 80 J = IHI + 1, N
        DO 70 I = 1, N
            A( I, J ) = ZERO
70    CONTINUE
        A( J, J ) = ONE
80 CONTINUE
*
    IF( NH.GT.0 ) THEN
*
*   Generate Q(ilo+1:ihl,ilo+1:ihl)

```



```

*
      CALL DORGQR( NH, NH, NH, A( ILO+1, ILO+1 ), LDA, TAU( ILO ),
$      WORK, LWORK, IINFO )
      END IF
      WORK( 1 ) = LWKOPT
      RETURN
*
*      End of DORGHR
*
      END

```

— LAPACK dorghr —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dorghr (n ilo ihi a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lwork lda ihi ilo n))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (iinfo 0) (j 0) (lwkopt 0) (nb 0) (nh 0) (lquery nil))
        (declare (type fixnum i iinfo j lwkopt nb nh)
                  (type (member t nil) lquery))
        (setf info 0)
        (setf nh (f2cl-lib:int-sub ihi ilo))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< n 0)
            (setf info -1))
          ((or (< ilo 1)
                (> ilo
                  (max (the fixnum 1) (the fixnum n))))
            (setf info -2))
          ((or
            (< ihi (min (the fixnum ilo) (the fixnum n)))
            (> ihi n))
            (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum n)))
            (setf info -5))
          ((and
            (< lwork
              (max (the fixnum 1) (the fixnum nh)))
            (not lquery))

```

```

      (setf info -8)))
(cond
  (= info 0)
    (setf nb (ilaenv 1 "DORGQR" " " nh nh nh -1))
    (setf lwkopt
      (f2cl-lib:int-mul
        (max (the fixnum 1) (the fixnum nh))
        nb))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum lwkopt) 'double-float))))
(cond
  (/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DORGHR" (f2cl-lib:int-sub info))
    (go end_label))
(lquery
  (go end_label)))
(cond
  (= n 0)
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum 1) 'double-float))
    (go end_label)))
(f2cl-lib:fdo (j ihi (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j (f2cl-lib:int-add ilo 1)) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%
        zero)))
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1) (f2cl-lib:int-add i 1))
        (> i ihi) nil)
        (tagbody
          (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
            (f2cl-lib:fref a-%data%
              (i (f2cl-lib:int-sub j 1))
              ((1 lda) (1 *))
              a-%offset%))))
          (f2cl-lib:fdo (i (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add i 1))
            (> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%
                zero))))))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j ilo) nil)
(tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
    (tagbody

```

```

      (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
            zero)))
    (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%)
          one)))
  (f2cl-lib:fdo (j (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)) a-%offset%)
            zero)))
      (setf (f2cl-lib:fref a-%data% (j j) ((1 lda) (1 *)) a-%offset%)
            one)))
  (cond
    (> nh 0)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (dorgqr nh nh nh
        (f2cl-lib:array-slice a
          double-float
          ((+ ilo 1) (f2cl-lib:int-add ilo 1))
          ((1 lda) (1 *)))
        lda (f2cl-lib:array-slice tau double-float (ilo) ((1 *))) work
        lwork iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7))
      (setf iinfo var-8))))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (coerce (the fixnum lwkopt) 'double-float))
end_label
  (return (values nil nil nil nil nil nil nil nil info))))

```

dorgl2 LAPACK

— dorgl2.input —

```

)set break resume
)sys rm -f dorgl2.output
)spool dorgl2.output
)set message test on
)set message auto off
)clear all

)spool

```

)lisp (bye)

— dorgl2.help —

=====

dorgl2 examples

=====

=====

Man Page Details

=====

NAME

DORGL2 - an m by n real matrix Q with orthonormal rows,

SYNOPSIS

SUBROUTINE DORGL2(M, N, K, A, LDA, TAU, WORK, INFO)

INTEGER INFO, K, LDA, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

Purpose

=====

DORGL2 generates an m by n real matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

$$Q = H(k) \dots H(2) H(1)$$

as returned by DGELQF.

Arguments

=====

M (input) INTEGER

The number of rows of the matrix Q. M >= 0.

N (input) INTEGER

The number of columns of the matrix Q. N >= M.

K (input) INTEGER

The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)

On entry, the i -th row must contain the vector which defines the elementary reflector $H(i)$, for $i = 1, 2, \dots, k$, as returned by DGELQF in the first k rows of its array argument A .
On exit, the m -by- n matrix Q .

LDA (input) INTEGER
The first dimension of the array A . $LDA \geq \max(1, M)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector $H(i)$, as returned by DGELQF.

WORK (workspace) DOUBLE PRECISION array, dimension (M)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-i$, the i -th argument has an illegal value

— dorgl2.f —

```

SUBROUTINE DORGL2( M, N, K, A, LDA, TAU, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      INTEGER          INFO, K, LDA, M, N
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      DOUBLE PRECISION  ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*  ..
*
*  .. Local Scalars ..
      INTEGER          I, J, L
*
*  ..
*
*  .. External Subroutines ..
      EXTERNAL          DLARF, DSCAL, XERBLA
*
*  ..

```

```

*    .. Intrinsic Functions ..
      INTRINSIC          MAX
*
*    ..
*    .. Executable Statements ..
*
*    Test the input arguments
*
      INFO = 0
      IF( M.LT.0 ) THEN
        INFO = -1
      ELSE IF( N.LT.M ) THEN
        INFO = -2
      ELSE IF( K.LT.0 .OR. K.GT.M ) THEN
        INFO = -3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
        INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORGL2', -INFO )
        RETURN
      END IF
*
*    Quick return if possible
*
      IF( M.LE.0 )
$    RETURN
*
      IF( K.LT.M ) THEN
*
*        Initialise rows k+1:m to rows of the unit matrix
*
        DO 20 J = 1, N
          DO 10 L = K + 1, M
            A( L, J ) = ZERO
10          CONTINUE
          IF( J.GT.K .AND. J.LE.M )
$            A( J, J ) = ONE
20          CONTINUE
        END IF
*
        DO 40 I = K, 1, -1
*
*        Apply H(i) to A(i:m,i:n) from the right
*
          IF( I.LT.N ) THEN
            IF( I.LT.M ) THEN
              A( I, I ) = ONE
              CALL DLARF( 'Right', M-I, N-I+1, A( I, I ), LDA,
$                TAU( I ), A( I+1, I ), LDA, WORK )
            END IF

```

```

        CALL DSCAL( N-I, -TAU( I ), A( I, I+1 ), LDA )
      END IF
      A( I, I ) = ONE - TAU( I )
*
*      Set A(i,1:i-1) to zero
*
      DO 30 L = 1, I - 1
        A( I, L ) = ZERO
30    CONTINUE
40 CONTINUE
      RETURN
*
*      End of DORGL2
*
      END

```

— LAPACK dorgl2 —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun dorgl2 (m n k a lda tau work info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lda k n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (j 0) (l 0))
        (declare (type fixnum i j l))
        (setf info 0)
        (cond
          ((< m 0)
            (setf info -1))
          ((< n m)
            (setf info -2))
          ((or (< k 0) (> k m))
            (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info -5)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DORGL2" (f2cl-lib:int-sub info))
            (go end_label)))

```

```

(if (<= m 0) (go end_label))
(cond
  (< k m)
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (l (f2cl-lib:int-add k 1) (f2cl-lib:int-add l 1))
      (> l m) nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data%
                          (l j)
                          ((1 lda) (1 *)))
        a-%offset%)
        zero)))
    (if (and (> j k) (<= j m))
      (setf (f2cl-lib:fref a-%data%
                          (j j)
                          ((1 lda) (1 *)))
        a-%offset%)
        one))))))
(f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
  (> i 1) nil)
(tagbody
  (cond
    (< i n)
    (cond
      (< i m)
      (setf (f2cl-lib:fref a-%data%
                          (i i)
                          ((1 lda) (1 *)))
        a-%offset%)
        one)
      (dlarf "Right" (f2cl-lib:int-sub m i)
        (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1)
        (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
        lda (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
        (f2cl-lib:array-slice a
          double-float
          ((+ i 1) i)
          ((1 lda) (1 *)))
        lda work)))
    (dscal (f2cl-lib:int-sub n i)
      (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
      (f2cl-lib:array-slice a
        double-float
        (i (f2cl-lib:int-add i 1))
        ((1 lda) (1 *)))
      lda)))
  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
    (- one

```



```

                (f2c1-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
(f2c1-lib:fdo (1 1 (f2c1-lib:int-add 1 1))
              (> 1 (f2c1-lib:int-add i (f2c1-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2c1-lib:fref a-%data% (i 1) ((1 lda) (1 *)) a-%offset%)
          zero))))
end_label
  (return (values nil nil nil nil nil nil nil info))))))

```

dorglq LAPACK

— dorglq.input —

```

)set break resume
)sys rm -f dorglq.output
)spool dorglq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorglq.help —

```

=====
dorglq examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DORGLQ - an M-by-N real matrix Q with orthonormal rows,

SYNOPSIS

SUBROUTINE DORGLQ(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)

INTEGER INFO, K, LDA, LWORK, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

Purpose
=====

DORGLQ generates an M-by-N real matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

$$Q = H(k) \dots H(2) H(1)$$

as returned by DGELQF.

Arguments
=====

M (input) INTEGER
The number of rows of the matrix Q. M >= 0.

N (input) INTEGER
The number of columns of the matrix Q. N >= M.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by DGELQF in the first k rows of its array argument A.
On exit, the M-by-N matrix Q.

LDA (input) INTEGER
The first dimension of the array A. LDA >= max(1,M).

TAU (input) DOUBLE PRECISION array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGELQF.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK >= max(1,M).
For optimum performance LWORK >= M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error

message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument has an illegal value

— dorglq.f —

```

SUBROUTINE DORGLQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    June 30, 1999
*
*    .. Scalar Arguments ..
      INTEGER          INFO, K, LDA, LWORK, M, N
*
*    ..
*
*    .. Array Arguments ..
      DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
*
*    ..
*
*  =====
*
*    .. Parameters ..
      DOUBLE PRECISION  ZERO
      PARAMETER         ( ZERO = 0.0D+0 )
*
*    ..
*
*    .. Local Scalars ..
      LOGICAL           LQUERY
      INTEGER           I, IB, IINFO, IWS, J, KI, KK, L, LDWORK,
$                      LWKOPT, NB, NBMIN, NX
*
*    ..
*
*    .. External Subroutines ..
      EXTERNAL          DLARFB, DLARFT, DORGL2, XERBLA
*
*    ..
*
*    .. Intrinsic Functions ..
      INTRINSIC         MAX, MIN
*
*    ..
*
*    .. External Functions ..
      INTEGER           ILAENV
      EXTERNAL          ILAENV
*
*    ..
*
*    .. Executable Statements ..
*
*    Test the input arguments

```

```

*
      INFO = 0
      NB = ILAENV( 1, 'DORGLQ', ' ', M, N, K, -1 )
      LWKOPT = MAX( 1, M )*NB
      WORK( 1 ) = LWKOPT
      LQUERY = ( LWORK.EQ.-1 )
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.M ) THEN
         INFO = -2
      ELSE IF( K.LT.0 .OR. K.GT.M ) THEN
         INFO = -3
      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
         INFO = -5
      ELSE IF( LWORK.LT.MAX( 1, M ) .AND. .NOT.LQUERY ) THEN
         INFO = -8
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DORGLQ', -INFO )
         RETURN
      ELSE IF( LQUERY ) THEN
         RETURN
      END IF

*
*      Quick return if possible
*
      IF( M.LE.0 ) THEN
         WORK( 1 ) = 1
         RETURN
      END IF

*
      NBMIN = 2
      NX = 0
      IWS = M
      IF( NB.GT.1 .AND. NB.LT.K ) THEN

*
*         Determine when to cross over from blocked to unblocked code.
*
      NX = MAX( 0, ILAENV( 3, 'DORGLQ', ' ', M, N, K, -1 ) )
      IF( NX.LT.K ) THEN

*
*         Determine if workspace is large enough for blocked code.
*
      LDWORK = M
      IWS = LDWORK*NB
      IF( LWORK.LT.IWS ) THEN

*
*         Not enough workspace to use optimal NB:  reduce NB and
*         determine the minimum value of NB.
*

```

```

        NB = LWORK / LDWORK
        NBMIN = MAX( 2, ILAENV( 2, 'DORGLQ', ' ', M, N, K, -1 ) )
    END IF
    END IF
END IF
*
IF( NB.GE.NBMIN .AND. NB.LT.K .AND. NX.LT.K ) THEN
*
    Use blocked code after the last block.
    The first kk rows are handled by the block method.
*
    KI = ( ( K-NX-1 ) / NB ) * NB
    KK = MIN( K, KI+NB )
*
    Set A(kk+1:m,1:kk) to zero.
*
    DO 20 J = 1, KK
        DO 10 I = KK + 1, M
            A( I, J ) = ZERO
10      CONTINUE
20      CONTINUE
    ELSE
        KK = 0
    END IF
*
    Use unblocked code for the last or only block.
*
    IF( KK.LT.M )
$      CALL DORGL2( M-KK, N-KK, K-KK, A( KK+1, KK+1 ), LDA,
$                  TAU( KK+1 ), WORK, IINFO )
*
    IF( KK.GT.0 ) THEN
*
        Use blocked code
*
        DO 50 I = KI + 1, 1, -NB
            IB = MIN( NB, K-I+1 )
            IF( I+IB.LE.M ) THEN
*
                Form the triangular factor of the block reflector
                H = H(i) H(i+1) . . . H(i+ib-1)
*
                CALL DLARFT( 'Forward', 'Rowwise', N-I+1, IB, A( I, I ),
$                  LDA, TAU( I ), WORK, LDWORK )
*
                Apply H' to A(i+ib:m,i:n) from the right
*
                CALL DLARFB( 'Right', 'Transpose', 'Forward', 'Rowwise',
$                  M-I-IB+1, N-I+1, IB, A( I, I ), LDA, WORK,
$                  LDWORK, A( I+IB, I ), LDA, WORK( IB+1 ),

```

```

$                                LDWORK )
    END IF
*
*    Apply H' to columns i:n of current block
*
    CALL DORGL2( IB, N-I+1, IB, A( I, I ), LDA, TAU( I ), WORK,
$              IINFO )
*
*    Set columns 1:i-1 of current block to zero
*
    DO 40 J = 1, I - 1
        DO 30 L = I, I + IB - 1
            A( L, J ) = ZERO
30        CONTINUE
40    CONTINUE
50    CONTINUE
    END IF
*
    WORK( 1 ) = IWS
    RETURN
*
*    End of DORGLQ
*
    END

```

— LAPACK dorglq —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dorglq (m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
              (type fixnum info lwork lda k n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ki 0) (kk 0) (l 0)
              (ldwork 0) (lwkopt 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type fixnum i ib iinfo iws j ki kk l ldwork
                          lwkopt nb nbmin nx)
                  (type (member t nil) lquery))
        (setf info 0)
        (setf nb (ilaenv 1 "DORGLQ" " " m n k -1))
        (setf lwkopt
          (f2cl-lib:int-mul
            (max (the fixnum 1) (the fixnum m))

```

```

        nb))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum lwkopt) 'double-float))
(setf lquery (coerce (= lwork -1) '(member t nil)))
(cond
  ((< m 0)
   (setf info -1))
  ((< n m)
   (setf info -2))
  ((or (< k 0) (> k m))
   (setf info -3))
  ((< lda (max (the fixnum 1) (the fixnum m)))
   (setf info -5))
  (and
   (< lwork (max (the fixnum 1) (the fixnum m)))
   (not lquery))
   (setf info -8)))
(cond
  (/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "DORGLQ" (f2cl-lib:int-sub info))
  (go end_label))
(lquery
 (go end_label)))
(cond
  ((<= m 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce (the fixnum 1) 'double-float))
   (go end_label)))
(setf nbmin 2)
(setf nx 0)
(setf iws m)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
        (max (the fixnum 0)
              (the fixnum
               (ilaenv 3 "DORGLQ" " " m n k -1)))))
  (cond
   ((< nx k)
    (setf ldwork m)
    (setf iws (f2cl-lib:int-mul ldwork nb))
    (cond
      ((< lwork iws)
       (setf nb (the fixnum (truncate lwork ldwork)))
       (setf nbmin
            (max (the fixnum 2)
                  (the fixnum
                   (ilaenv 2 "DORGLQ" " " m n k -1))))))))))

```

```

(cond
  ((and (>= nb nbmin) (< nb k) (< nx k))
   (setf ki (* (the fixnum (truncate (- k nx 1) nb)) nb))
   (setf kk
        (min (the fixnum k)
              (the fixnum (f2cl-lib:int-add ki nb)))))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                ((> j kk) nil)
    (tagbody
      (f2cl-lib:fdo (i (f2cl-lib:int-add kk 1) (f2cl-lib:int-add i 1))
                    ((> i m) nil)
        (tagbody
          (setf (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *)))
                a-%offset%
                zero))))))
  (t
   (setf kk 0)))
(if (< kk m)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
      (dorgl2 (f2cl-lib:int-sub m kk) (f2cl-lib:int-sub n kk)
              (f2cl-lib:int-sub k kk)
              (f2cl-lib:array-slice a
                                     double-float
                                     ((+ kk 1) (f2cl-lib:int-add kk 1))
                                     ((1 lda) (1 *)))
              lda (f2cl-lib:array-slice tau double-float ((+ kk 1)) ((1 *)))
              work iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
      (setf iinfo var-7)))
(cond
  ((> kk 0)
   (f2cl-lib:fdo (i (f2cl-lib:int-add ki 1)
                    (f2cl-lib:int-add i (f2cl-lib:int-sub nb)))
                 ((> i 1) nil)
     (tagbody
      (setf ib
              (min (the fixnum nb)
                    (the fixnum
                     (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
      (cond
        ((<= (f2cl-lib:int-add i ib) m)
         (dlarft "Forward" "Rowwise"
                  (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
                  (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
                  lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
                  ldwork)
         (dlarfb "Right" "Transpose" "Forward" "Rowwise"

```



```

(f2cl-lib:int-add (f2cl-lib:int-sub m i ib) 1)
(f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
(f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
lda work ldwork
(f2cl-lib:array-slice a
                        double-float
                        ((+ i ib) i)
                        ((1 lda) (1 *)))

lda
(f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
ldwork)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
  (dorgl2 ib (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1) ib
           (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
           lda (f2cl-lib:array-slice tau double-float (i) ((1 *)))
           work iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
  (setf iinfo var-7))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              ((> j (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
               nil)
  (tagbody
    (f2cl-lib:fdo (l i (f2cl-lib:int-add l 1))
                  ((> l
                     (f2cl-lib:int-add i
                                       ib
                                       (f2cl-lib:int-sub 1)))
                    nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
                              (l j)
                              ((1 lda) (1 *))
                              a-%offset%)
              zero))))))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum iws) 'double-float))
end_label
(return (values nil nil nil nil nil nil nil nil info))))

```

dorgqr LAPACK

— dorgqr.input —

```

)set break resume
)sys rm -f dorgqr.output
)spool dorgqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorgqr.help —

=====

dorgqr examples

=====

=====

Man Page Details

=====

NAME

DORGQR - an M-by-N real matrix Q with orthonormal columns,

SYNOPSIS

SUBROUTINE DORGQR(M, N, K, A, LDA, TAU, WORK, LWORK, INFO)

INTEGER INFO, K, LDA, LWORK, M, N

DOUBLE PRECISION A(LDA, *), TAU(*), WORK(*)

Purpose

=====

DORGQR generates an M-by-N real matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF.

Arguments

=====

M (input) INTEGER

The number of rows of the matrix Q. M >= 0.

N (input) INTEGER
The number of columns of the matrix Q. $M \geq N \geq 0$.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. $N \geq K \geq 0$.

A (input/output) DOUBLE PRECISION array, dimension (LDA,N)
On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for $i = 1, 2, \dots, k$, as returned by DGEQRF in the first k columns of its array argument A.
On exit, the M-by-N matrix Q.

LDA (input) INTEGER
The first dimension of the array A. $LDA \geq \max(1, M)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1, N)$.
For optimum performance $LWORK \geq N \cdot NB$, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument has an illegal value

— dorgqr.f —

```

SUBROUTINE DORGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
*
* -- LAPACK routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   June 30, 1999

```

```

*
*   .. Scalar Arguments ..
*   INTEGER          INFO, K, LDA, LWORK, M, N
*   ..
*   .. Array Arguments ..
*   DOUBLE PRECISION  A( LDA, * ), TAU( * ), WORK( * )
*   ..
*
*   =====
*
*   .. Parameters ..
*   DOUBLE PRECISION  ZERO
*   PARAMETER         ( ZERO = 0.0D+0 )
*   ..
*   .. Local Scalars ..
*   LOGICAL           LQUERY
*   INTEGER           I, IB, IINFO, IWS, J, KI, KK, L, LDWORK,
*   $                 LWKOPT, NB, NBMIN, NX
*   ..
*   .. External Subroutines ..
*   EXTERNAL          DLARFB, DLARFT, DORG2R, XERBLA
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC         MAX, MIN
*   ..
*   .. External Functions ..
*   INTEGER           ILAENV
*   EXTERNAL          ILAENV
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
*   INFO = 0
*   NB = ILAENV( 1, 'DORGQR', ' ', M, N, K, -1 )
*   LWKOPT = MAX( 1, N )*NB
*   WORK( 1 ) = LWKOPT
*   LQUERY = ( LWORK.EQ.-1 )
*   IF( M.LT.0 ) THEN
*       INFO = -1
*   ELSE IF( N.LT.0 .OR. N.GT.M ) THEN
*       INFO = -2
*   ELSE IF( K.LT.0 .OR. K.GT.N ) THEN
*       INFO = -3
*   ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
*       INFO = -5
*   ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
*       INFO = -8
*   END IF
*   IF( INFO.NE.0 ) THEN

```

```

        CALL XERBLA( 'DORGQR', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF
*
*   Quick return if possible
*
*
    IF( N.LE.0 ) THEN
        WORK( 1 ) = 1
        RETURN
    END IF
*
    NBMIN = 2
    NX = 0
    IWS = N
    IF( NB.GT.1 .AND. NB.LT.K ) THEN
*
*       Determine when to cross over from blocked to unblocked code.
*
*
        NX = MAX( 0, ILAENV( 3, 'DORGQR', ' ', M, N, K, -1 ) )
        IF( NX.LT.K ) THEN
*
*           Determine if workspace is large enough for blocked code.
*
*
            LDWORK = N
            IWS = LDWORK*NB
            IF( LWORK.LT.IWS ) THEN
*
*               Not enough workspace to use optimal NB:  reduce NB and
*               determine the minimum value of NB.
*
*
                NB = LWORK / LDWORK
                NBMIN = MAX( 2, ILAENV( 2, 'DORGQR', ' ', M, N, K, -1 ) )
            END IF
        END IF
    END IF
*
    IF( NB.GE.NBMIN .AND. NB.LT.K .AND. NX.LT.K ) THEN
*
*       Use blocked code after the last block.
*       The first kk columns are handled by the block method.
*
*
        KI = ( ( K-NX-1 ) / NB ) * NB
        KK = MIN( K, KI+NB )
*
*       Set A(1:kk,kk+1:n) to zero.
*
*
        DO 20 J = KK + 1, N
            DO 10 I = 1, KK

```

```

          A( I, J ) = ZERO
10      CONTINUE
20      CONTINUE
      ELSE
          KK = 0
      END IF
*
*      Use unblocked code for the last or only block.
*
      IF( KK.LT.N )
$      CALL DORG2R( M-KK, N-KK, K-KK, A( KK+1, KK+1 ), LDA,
$                  TAU( KK+1 ), WORK, IINFO )
*
      IF( KK.GT.0 ) THEN
*
*      Use blocked code
*
      DO 50 I = KI + 1, 1, -NB
          IB = MIN( NB, K-I+1 )
          IF( I+IB.LE.N ) THEN
*
*      Form the triangular factor of the block reflector
*      H = H(i) H(i+1) . . . H(i+ib-1)
*
          CALL DLARFT( 'Forward', 'Columnwise', M-I+1, IB,
$                  A( I, I ), LDA, TAU( I ), WORK, LDWORK )
*
*      Apply H to A(i:m,i+ib:n) from the left
*
          CALL DLARFB( 'Left', 'No transpose', 'Forward',
$                  'Columnwise', M-I+1, N-I-IB+1, IB,
$                  A( I, I ), LDA, WORK, LDWORK, A( I, I+IB ),
$                  LDA, WORK( IB+1 ), LDWORK )
          END IF
*
*      Apply H to rows i:m of current block
*
          CALL DORG2R( M-I+1, IB, IB, A( I, I ), LDA, TAU( I ), WORK,
$                  IINFO )
*
*      Set rows 1:i-1 of current block to zero
*
      DO 40 J = I, I + IB - 1
          DO 30 L = 1, I - 1
              A( L, J ) = ZERO
          30      CONTINUE
          40      CONTINUE
          50      CONTINUE
      END IF
*

```

```

      WORK( 1 ) = IWS
      RETURN
*
*      End of DORGQR
*
      END

```

— LAPACK dorgqr —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dorgqr (m n k a lda tau work lwork info)
    (declare (type (simple-array double-float (*)) work tau a)
      (type fixnum info lwork lda k n m))
    (f2cl-lib:with-multi-array-data
      ((a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ki 0) (kk 0) (l 0)
        (ldwork 0) (lwkopt 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare (type fixnum i ib iinfo iws j ki kk l ldwork
          lwkopt nb nbmin nx)
          (type (member t nil) lquery))
        (setf info 0)
        (setf nb (ilaenv 1 "DORGQR" " " m n k -1))
        (setf lwkopt
          (f2cl-lib:int-mul
            (max (the fixnum 1) (the fixnum n))
            nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce (the fixnum lwkopt) 'double-float))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          ((< m 0)
            (setf info -1))
          ((or (< n 0) (> n m))
            (setf info -2))
          ((or (< k 0) (> k n))
            (setf info -3))
          ((< lda (max (the fixnum 1) (the fixnum m)))
            (setf info -5))
          ((and
            (< lwork (max (the fixnum 1) (the fixnum n)))
            (not lquery))
            (setf info -8)))
        (cond

```

```

( (/ = info 0)
  (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DORGQR" (f2cl-lib:int-sub info))
  (go end_label))
(lquery
  (go end_label)))
(cond
  ((<= n 0)
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum 1) 'double-float))
    (go end_label)))
(setf nbmin 2)
(setf nx 0)
(setf iws n)
(cond
  ((and (> nb 1) (< nb k))
    (setf nx
      (max (the fixnum 0)
        (the fixnum
          (ilaenv 3 "DORGQR" " " m n k -1)))))
  (cond
    ((< nx k)
      (setf ldwork n)
      (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
          (setf nb (the fixnum (truncate lwork ldwork)))
          (setf nbmin
            (max (the fixnum 2)
              (the fixnum
                (ilaenv 2 "DORGQR" " " m n k -1))))))))))
  (cond
    ((and (>= nb nbmin) (< nb k) (< nx k))
      (setf ki (* (the fixnum (truncate (- k nx 1) nb)) nb))
      (setf kk
        (min (the fixnum k)
          (the fixnum (f2cl-lib:int-add ki nb))))
      (f2cl-lib:fdo (j (f2cl-lib:int-add kk 1) (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i kk) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data%
                (i j)
                ((1 lda) (1 *))
                a-%offset%)
                zero))))))
  (t

```



```

(setf kk 0)))
(if (< kk n)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (dorg2r (f2cl-lib:int-sub m kk) (f2cl-lib:int-sub n kk)
      (f2cl-lib:int-sub k kk)
      (f2cl-lib:array-slice a
        double-float
        ((+ kk 1) (f2cl-lib:int-add kk 1))
        ((1 lda) (1 *)))
      lda (f2cl-lib:array-slice tau double-float ((+ kk 1)) ((1 *)))
      work iinfo)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))
    (setf iinfo var-7)))
(cond
  ((> kk 0)
    (f2cl-lib:fdo (i (f2cl-lib:int-add ki 1)
      (f2cl-lib:int-add i (f2cl-lib:int-sub nb)))
      ((> i 1) nil)
    (tagbody
      (setf ib
        (min (the fixnum nb)
          (the fixnum
            (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
      (cond
        ((<= (f2cl-lib:int-add i ib) n)
          (dlarft "Forward" "Columnwise"
            (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
            lda (f2cl-lib:array-slice tau double-float (i) ((1 *))) work
            ldwork)
          (dlarfb "Left" "No transpose" "Forward" "Columnwise"
            (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
            (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
            lda work ldwork
            (f2cl-lib:array-slice a
              double-float
              (i (f2cl-lib:int-add i ib))
              ((1 lda) (1 *)))
            lda
            (f2cl-lib:array-slice work double-float ((+ ib 1)) ((1 *)))
            ldwork)))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
          (dorg2r (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib ib
            (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *)))
            lda (f2cl-lib:array-slice tau double-float (i) ((1 *)))
            work iinfo)
          (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6))

```

```

      (setf iinfo var-7))
    (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
      (> j
        (f2cl-lib:int-add i ib (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (f2cl-lib:fdo (l 1 (f2cl-lib:int-add l 1))
        (> l
          (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
          nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data%
                           (l j)
                           ((1 lda) (1 *))
                           a-%offset%)
              zero))))))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum iws) 'double-float))
  end_label
  (return (values nil nil nil nil nil nil nil info))))

```

dorm2r LAPACK

— dorm2r.input —

```

)set break resume
)sys rm -f dorm2r.output
)spool dorm2r.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorm2r.help —

```

=====
dorm2r examples
=====

```

```
=====
Man Page Details
=====
```

NAME

DORM2R - the general real m by n matrix C with $Q * C$ if $SIDE = 'L'$ and $TRANS = 'N'$, or $Q' * C$ if $SIDE = 'L'$ and $TRANS = 'T'$, or $C * Q$ if $SIDE = 'R'$ and $TRANS = 'N'$, or $C * Q'$ if $SIDE = 'R'$ and $TRANS = 'T'$,

SYNOPSIS

```
SUBROUTINE DORM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  INFO )
```

```
CHARACTER      SIDE, TRANS
```

```
INTEGER        INFO, K, LDA, LDC, M, N
```

```
DOUBLE         PRECISION A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
* )
```

Purpose

```
=====
```

DORM2R overwrites the general real m by n matrix C with

$Q * C$ if $SIDE = 'L'$ and $TRANS = 'N'$, or

$Q' * C$ if $SIDE = 'L'$ and $TRANS = 'T'$, or

$C * Q$ if $SIDE = 'R'$ and $TRANS = 'N'$, or

$C * Q'$ if $SIDE = 'R'$ and $TRANS = 'T'$,

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF. Q is of order m if $SIDE = 'L'$ and of order n if $SIDE = 'R'$.

Arguments

```
=====
```

SIDE (input) CHARACTER*1
 = 'L': apply Q or Q' from the Left
 = 'R': apply Q or Q' from the Right

TRANS (input) CHARACTER*1

```

= 'N': apply Q (No transpose)
= 'T': apply Q' (Transpose)

M      (input) INTEGER
       The number of rows of the matrix C. M >= 0.

N      (input) INTEGER
       The number of columns of the matrix C. N >= 0.

K      (input) INTEGER
       The number of elementary reflectors whose product defines
       the matrix Q.
       If SIDE = 'L', M >= K >= 0;
       if SIDE = 'R', N >= K >= 0.

A      (input) DOUBLE PRECISION array, dimension (LDA,K)
       The i-th column must contain the vector which defines the
       elementary reflector H(i), for i = 1,2,...,k, as returned by
       DGEQRF in the first k columns of its array argument A.
       A is modified by the routine but restored on exit.

LDA     (input) INTEGER
       The leading dimension of the array A.
       If SIDE = 'L', LDA >= max(1,M);
       if SIDE = 'R', LDA >= max(1,N).

TAU     (input) DOUBLE PRECISION array, dimension (K)
       TAU(i) must contain the scalar factor of the elementary
       reflector H(i), as returned by DGEQRF.

C      (input/output) DOUBLE PRECISION array, dimension (LDC,N)
       On entry, the m by n matrix C.
       On exit, C is overwritten by Q*C or Q'*C or C*Q' or C*Q.

LDC     (input) INTEGER
       The leading dimension of the array C. LDC >= max(1,M).

WORK    (workspace) DOUBLE PRECISION array, dimension
              (N) if SIDE = 'L',
              (M) if SIDE = 'R'

INFO    (output) INTEGER
       = 0: successful exit
       < 0: if INFO = -i, the i-th argument had an illegal value

```

```

      SUBROUTINE DORM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
$      WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  February 29, 1992
*
*  .. Scalar Arguments ..
*  CHARACTER          SIDE, TRANS
*  INTEGER            INFO, K, LDA, LDC, M, N
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION   A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*  ..
*
*  =====
*
*  .. Parameters ..
*  DOUBLE PRECISION   ONE
*  PARAMETER          ( ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL            LEFT, NOTRAN
*  INTEGER            I, I1, I2, I3, IC, JC, MI, NI, NQ
*  DOUBLE PRECISION   AII
*
*  ..
*  .. External Functions ..
*  LOGICAL            LSAME
*  EXTERNAL           LSAME
*
*  ..
*  .. External Subroutines ..
*  EXTERNAL           DLARF, XERBLA
*
*  ..
*  .. Intrinsic Functions ..
*  INTRINSIC          MAX
*
*  ..
*  .. Executable Statements ..
*
*  Test the input arguments
*
*
*  INFO = 0
*  LEFT = LSAME( SIDE, 'L' )
*  NOTRAN = LSAME( TRANS, 'N' )
*
*
*  NQ is the order of Q
*
*
*  IF( LEFT ) THEN
*     NQ = M
*  ELSE

```

```

      NQ = N
      END IF
      IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
        INFO = -1
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) ) THEN
        INFO = -2
      ELSE IF( M.LT.0 ) THEN
        INFO = -3
      ELSE IF( N.LT.0 ) THEN
        INFO = -4
      ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
        INFO = -5
      ELSE IF( LDA.LT.MAX( 1, NQ ) ) THEN
        INFO = -7
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
        INFO = -10
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORM2R', -INFO )
        RETURN
      END IF
*
*   Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 )
$      RETURN
*
      IF( ( LEFT .AND. .NOT.NOTRAN ) .OR. ( .NOT.LEFT .AND. NOTRAN ) )
$      THEN
        I1 = 1
        I2 = K
        I3 = 1
      ELSE
        I1 = K
        I2 = 1
        I3 = -1
      END IF
*
      IF( LEFT ) THEN
        NI = N
        JC = 1
      ELSE
        MI = M
        IC = 1
      END IF
*
      DO 10 I = I1, I2, I3
        IF( LEFT ) THEN
*
*           H(i) is applied to C(i:m,1:n)

```

```

*
      MI = M - I + 1
      IC = I
    ELSE
*
*      H(i) is applied to C(1:m,i:n)
*
      NI = N - I + 1
      JC = I
    END IF
*
*      Apply H(i)
*
      AII = A( I, I )
      A( I, I ) = ONE
      CALL DLARF( SIDE, MI, NI, A( I, I ), 1, TAU( I ), C( IC, JC ),
$          LDC, WORK )
      A( I, I ) = AII
10 CONTINUE
    RETURN
*
*      End of DORM2R
*
    END

```

— LAPACK dorm2r —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dorm2r (side trans m n k a lda tau c ldc work info)
    (declare (type (simple-array double-float (*)) work c tau a)
      (type fixnum info ldc lda k n m)
      (type character trans side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((aii 0.0) (i 0) (i1 0) (i2 0) (i3 0) (ic 0) (jc 0) (mi 0) (ni 0)
        (nq 0) (left nil) (notran nil))
        (declare (type (double-float) aii)
          (type fixnum i i1 i2 i3 ic jc mi ni nq)
          (type (member t nil) left notran))
        (setf info 0)

```

```

(setf left (char-equal side #\L))
(setf notran (char-equal trans #\N))
(cond
  (left
    (setf nq m))
  (t
    (setf nq n)))
(cond
  ((and (not left) (not (char-equal side #\R)))
    (setf info -1))
  ((and (not notran) (not (char-equal trans #\T)))
    (setf info -2))
  ((< m 0)
    (setf info -3))
  ((< n 0)
    (setf info -4))
  ((or (< k 0) (> k nq))
    (setf info -5))
  ((< lda (max (the fixnum 1) (the fixnum nq)))
    (setf info -7))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
    (setf info -10)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DORM2R" (f2cl-lib:int-sub info))
    (go end_label)))
(if (or (= m 0) (= n 0) (= k 0)) (go end_label))
(cond
  ((or (and left (not notran)) (and (not left) notran))
    (setf i1 1)
    (setf i2 k)
    (setf i3 1))
  (t
    (setf i1 k)
    (setf i2 1)
    (setf i3 -1)))
(cond
  (left
    (setf ni n)
    (setf jc 1))
  (t
    (setf mi m)
    (setf ic 1)))
(f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
  (> i i2) nil)
(tagbody
  (cond
    (left

```



```

        (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
        (setf ic i))
      (t
        (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
        (setf jc i)))
      (setf aii
        (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%))
      (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        one)
      (dlarf side mi ni
        (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) 1
        (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
        (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *))) ldc
        work)
      (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
        aii)))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil info))))))

```

dormbr LAPACK

— dormbr.input —

```

)set break resume
)sys rm -f dormbr.output
)spool dormbr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dormbr.help —

```

=====
dormbr examples
=====

```

```

=====
Man Page Details

```

```
=====
NAME
    DORMBR - = 'Q', DORMBR overwrites the general real M-by-N matrix C with
    SIDE = 'L' SIDE = 'R' TRANS = 'N'

SYNOPSIS
    SUBROUTINE DORMBR( VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
                      WORK, LWORK, INFO )

        CHARACTER          SIDE, TRANS, VECT

        INTEGER            INFO, K, LDA, LDC, LWORK, M, N

        DOUBLE             PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK(
            * )

Purpose
=====

If VECT = 'Q', DORMBR overwrites the general real M-by-N matrix C
with
            SIDE = 'L'      SIDE = 'R'
TRANS = 'N':      Q * C      C * Q
TRANS = 'T':      Q**T * C    C * Q**T

If VECT = 'P', DORMBR overwrites the general real M-by-N matrix C
with
            SIDE = 'L'      SIDE = 'R'
TRANS = 'N':      P * C      C * P
TRANS = 'T':      P**T * C    C * P**T

Here Q and P**T are the orthogonal matrices determined by DGEBRD when
reducing a real matrix A to bidiagonal form:  $A = Q * B * P^{**T}$ . Q and
P**T are defined as products of elementary reflectors H(i) and G(i)
respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the
order of the orthogonal matrix Q or P**T that is applied.

If VECT = 'Q', A is assumed to have been an NQ-by-K matrix:
if nq >= k, Q = H(1) H(2) . . . H(k);
if nq < k, Q = H(1) H(2) . . . H(nq-1).

If VECT = 'P', A is assumed to have been a K-by-NQ matrix:
if k < nq, P = G(1) G(2) . . . G(k);
if k >= nq, P = G(1) G(2) . . . G(nq-1).

Arguments
=====
```

VECT (input) CHARACTER*1
 = 'Q': apply Q or Q**T;
 = 'P': apply P or P**T.

SIDE (input) CHARACTER*1
 = 'L': apply Q, Q**T, P or P**T from the Left;
 = 'R': apply Q, Q**T, P or P**T from the Right.

TRANS (input) CHARACTER*1
 = 'N': No transpose, apply Q or P;
 = 'T': Transpose, apply Q**T or P**T.

M (input) INTEGER
 The number of rows of the matrix C. $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix C. $N \geq 0$.

K (input) INTEGER
 If VECT = 'Q', the number of columns in the original matrix reduced by DGEHRD.
 If VECT = 'P', the number of rows in the original matrix reduced by DGEHRD.
 $K \geq 0$.

A (input) DOUBLE PRECISION array, dimension
 (LDA,min(nq,K)) if VECT = 'Q'
 (LDA,nq) if VECT = 'P'
 The vectors which define the elementary reflectors H(i) and G(i), whose products determine the matrices Q and P, as returned by DGEHRD.

LDA (input) INTEGER
 The leading dimension of the array A.
 If VECT = 'Q', $LDA \geq \max(1,nq)$;
 if VECT = 'P', $LDA \geq \max(1,\min(nq,K))$.

TAU (input) DOUBLE PRECISION array, dimension (min(nq,K))
 TAU(i) must contain the scalar factor of the elementary reflector H(i) or G(i) which determines Q or P, as returned by DGEHRD in the array argument TAUQ or TAUP.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the M-by-N matrix C.
 On exit, C is overwritten by Q*C or Q**T*C or C*Q**T or C*Q or P*C or P**T*C or C*P or C*P**T.

LDC (input) INTEGER
 The leading dimension of the array C. $LDC \geq \max(1,M)$.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If SIDE = 'L', LWORK $\geq \max(1, N)$;
if SIDE = 'R', LWORK $\geq \max(1, M)$.
For optimum performance LWORK $\geq N \cdot \text{NB}$ if SIDE = 'L', and
LWORK $\geq M \cdot \text{NB}$ if SIDE = 'R', where NB is the optimal
blocksize.

If LWORK = -1, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

— dormbr.f —

```

      SUBROUTINE DORMBR( VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C,
$                      LDC, WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
*  CHARACTER            SIDE, TRANS, VECT
*  INTEGER              INFO, K, LDA, LDC, LWORK, M, N
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION     A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*  ..
*  =====
*
*  .. Local Scalars ..
*  LOGICAL              APPLYQ, LEFT, LQUERY, NOTRAN
*  CHARACTER            TRANST
*  INTEGER              I1, I2, IINFO, LWKOPT, MI, NB, NI, NQ, NW
*  ..

```

```

*      .. External Functions ..
LOGICAL          LSAME
INTEGER          ILAENV
EXTERNAL         LSAME, ILAENV
*
*      ..
*      .. External Subroutines ..
EXTERNAL         DORMLQ, DORMQR, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC        MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      APPLYQ = LSAME( VECT, 'Q' )
      LEFT = LSAME( SIDE, 'L' )
      NOTRAN = LSAME( TRANS, 'N' )
      LQUERY = ( LWORK.EQ.-1 )
*
*      NQ is the order of Q or P and NW is the minimum dimension of WORK
*
      IF( LEFT ) THEN
        NQ = M
        NW = N
      ELSE
        NQ = N
        NW = M
      END IF
      IF( .NOT.APPLYQ .AND. .NOT.LSAME( VECT, 'P' ) ) THEN
        INFO = -1
      ELSE IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
        INFO = -2
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) ) THEN
        INFO = -3
      ELSE IF( M.LT.0 ) THEN
        INFO = -4
      ELSE IF( N.LT.0 ) THEN
        INFO = -5
      ELSE IF( K.LT.0 ) THEN
        INFO = -6
      ELSE IF( ( APPLYQ .AND. LDA.LT.MAX( 1, NQ ) ) .OR.
$          ( .NOT.APPLYQ .AND. LDA.LT.MAX( 1, MIN( NQ, K ) ) ) )
$          THEN
        INFO = -8
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
        INFO = -11
      ELSE IF( LWORK.LT.MAX( 1, NW ) .AND. .NOT.LQUERY ) THEN
        INFO = -13

```

```

      END IF
*
      IF( INFO.EQ.0 ) THEN
        IF( APPLYQ ) THEN
          IF( LEFT ) THEN
            NB = ILAENV( 1, 'DORMQR', SIDE // TRANS, M-1, N, M-1,
$              -1 )
          ELSE
            NB = ILAENV( 1, 'DORMQR', SIDE // TRANS, M, N-1, N-1,
$              -1 )
          END IF
        ELSE
          IF( LEFT ) THEN
            NB = ILAENV( 1, 'DORMLQ', SIDE // TRANS, M-1, N, M-1,
$              -1 )
          ELSE
            NB = ILAENV( 1, 'DORMLQ', SIDE // TRANS, M, N-1, N-1,
$              -1 )
          END IF
        END IF
        LWKOPT = MAX( 1, NW )*NB
        WORK( 1 ) = LWKOPT
      END IF
*
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORMBR', -INFO )
        RETURN
      ELSE IF( LQUERY ) THEN
        RETURN
      END IF
*
*      Quick return if possible
*
      WORK( 1 ) = 1
      IF( M.EQ.0 .OR. N.EQ.0 )
$        RETURN
*
      IF( APPLYQ ) THEN
*
*      Apply Q
*
        IF( NQ.GE.K ) THEN
*
*      Q was determined by a call to DGEBRD with nq >= k
*
          CALL DORMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
$            WORK, LWORK, IINFO )
        ELSE IF( NQ.GT.1 ) THEN
*
*      Q was determined by a call to DGEBRD with nq < k

```

```

*
      IF( LEFT ) THEN
        MI = M - 1
        NI = N
        I1 = 2
        I2 = 1
      ELSE
        MI = M
        NI = N - 1
        I1 = 1
        I2 = 2
      END IF
      CALL DORMQR( SIDE, TRANS, MI, NI, NQ-1, A( 2, 1 ), LDA, TAU,
$      C( I1, I2 ), LDC, WORK, LWORK, IINFO )
      END IF
    ELSE
*
*      Apply P
*
      IF( NOTRAN ) THEN
        TRANST = 'T'
      ELSE
        TRANST = 'N'
      END IF
      IF( NQ.GT.K ) THEN
*
*      P was determined by a call to DGEHRD with nq > k
*
        CALL DORMLQ( SIDE, TRANST, M, N, K, A, LDA, TAU, C, LDC,
$      WORK, LWORK, IINFO )
      ELSE IF( NQ.GT.1 ) THEN
*
*      P was determined by a call to DGEHRD with nq <= k
*
        IF( LEFT ) THEN
          MI = M - 1
          NI = N
          I1 = 2
          I2 = 1
        ELSE
          MI = M
          NI = N - 1
          I1 = 1
          I2 = 2
        END IF
        CALL DORMLQ( SIDE, TRANST, MI, NI, NQ-1, A( 1, 2 ), LDA,
$      TAU, C( I1, I2 ), LDC, WORK, LWORK, IINFO )
      END IF
    END IF
    WORK( 1 ) = LWKOPT

```

```

      RETURN
*
*      End of DORMBR
*
      END

```

— LAPACK dormbr —

```

(defun dormbr (vect side trans m n k a lda tau c ldc work lwork info)
  (declare (type (simple-array double-float (*)) work c tau a)
            (type fixnum info lwork ldc lda k n m)
            (type character trans side vect))
  (f2cl-lib:with-multi-array-data
    ((vect character vect-%data% vect-%offset%)
     (side character side-%data% side-%offset%)
     (trans character trans-%data% trans-%offset%)
     (a double-float a-%data% a-%offset%)
     (tau double-float tau-%data% tau-%offset%)
     (c double-float c-%data% c-%offset%)
     (work double-float work-%data% work-%offset%))
    (prog ((i1 0) (i2 0) (iinfo 0) (lwkopt 0) (mi 0) (nb 0) (ni 0) (nq 0)
           (nw 0)
           (transt
            (make-array '(1) :element-type 'character :initial-element #\ ))
            (applyq nil) (left nil) (lquery nil) (notran nil))
           (declare (type (member t nil) notran lquery left applyq)
                     (type (simple-array character (1)) transt)
                     (type fixnum nw nq ni nb mi lwkopt iinfo i2 i1))
           (setf info 0)
           (setf applyq (char-equal vect #\Q))
           (setf left (char-equal side #\L))
           (setf notran (char-equal trans #\N))
           (setf lquery (coerce (= lwork -1) '(member t nil)))
           (cond
            (left
             (setf nq m)
             (setf nw n))
            (t
             (setf nq n)
             (setf nw m)))
           (cond
            ((and (not applyq) (not (char-equal vect #\P)))
             (setf info -1))
            ((and (not left) (not (char-equal side #\R)))
             (setf info -2))
            ((and (not notran) (not (char-equal trans #\T)))

```



```

      (setf info -3))
    ((< m 0)
      (setf info -4))
    ((< n 0)
      (setf info -5))
    ((< k 0)
      (setf info -6))
    (or
      (and applyq
        (< lda
          (max (the fixnum 1) (the fixnum nq))))
      (and (not applyq)
        (< lda
          (max (the fixnum 1)
            (the fixnum
              (min (the fixnum nq)
                (the fixnum k))))))))
    (setf info -8))
    ((< ldc (max (the fixnum 1) (the fixnum m)))
      (setf info -11))
    ((and
      (< lwork (max (the fixnum 1) (the fixnum nw)))
      (not lquery))
      (setf info -13)))
  (cond
    ((= info 0)
      (cond
        (applyq
          (cond
            (left
              (setf nb
                (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans)
                  (f2cl-lib:int-sub m 1) n (f2cl-lib:int-sub m 1) -1)))
            (t
              (setf nb
                (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans) m
                  (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) -1))))
          (t
            (cond
              (left
                (setf nb
                  (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans)
                    (f2cl-lib:int-sub m 1) n (f2cl-lib:int-sub m 1) -1)))
              (t
                (setf nb
                  (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans) m
                    (f2cl-lib:int-sub n 1) (f2cl-lib:int-sub n 1) -1))))))
      (setf lwkopt
        (f2cl-lib:int-mul
          (max (the fixnum 1) (the fixnum nw))

```

```

        nb))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce (the fixnum lwkopt) 'double-float)))
    (cond
      ((/= info 0)
        (error
          " ** On entry to ~a parameter number ~a had an illegal value~%"
          "DORMBR" (f2cl-lib:int-sub info))
        (go end_label))
      (lquery
        (go end_label)))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce (the fixnum 1) 'double-float))
    (if (or (= m 0) (= n 0)) (go end_label))
    (cond
      (applyq
        (cond
          ((>= nq k)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
               var-10 var-11 var-12)
              (dormqr side trans m n k a lda tau c ldc work lwork iinfo)
            (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                          var-8 var-9 var-10 var-11))
            (setf iinfo var-12))))
          (> nq 1)
            (cond
              (left
                (setf mi (f2cl-lib:int-sub m 1))
                (setf ni n)
                (setf i1 2)
                (setf i2 1))
              (t
                (setf mi m)
                (setf ni (f2cl-lib:int-sub n 1))
                (setf i1 1)
                (setf i2 2)))
              (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
                 var-10 var-11 var-12)
                (dormqr side trans mi ni (f2cl-lib:int-sub nq 1)
                  (f2cl-lib:array-slice a double-float (2 1) ((1 lda) (1 *)))
                  lda tau
                  (f2cl-lib:array-slice c double-float (i1 i2) ((1 ldc) (1 *)))
                  ldc work lwork iinfo)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                            var-8 var-9 var-10 var-11))
              (setf iinfo var-12))))))
      (t
        (cond

```

```

(notran
  (f2cl-lib:f2cl-set-string transt "T" (string 1)))
(t
  (f2cl-lib:f2cl-set-string transt "N" (string 1))))
(cond
  (> nq k)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
     var-10 var-11 var-12)
    (dormlq side transt m n k a lda tau c ldc work lwork iinfo)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                     var-8 var-9 var-10 var-11))
    (setf iinfo var-12)))
  (> nq 1)
  (cond
    (left
      (setf mi (f2cl-lib:int-sub m 1))
      (setf ni n)
      (setf i1 2)
      (setf i2 1))
    (t
      (setf mi m)
      (setf ni (f2cl-lib:int-sub n 1))
      (setf i1 1)
      (setf i2 2)))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
     var-10 var-11 var-12)
    (dormlq side transt mi ni (f2cl-lib:int-sub nq 1)
      (f2cl-lib:array-slice a double-float (1 2) ((1 lda) (1 *)))
      lda tau
      (f2cl-lib:array-slice c double-float (i1 i2) ((1 ldc) (1 *)))
      ldc work lwork iinfo)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                     var-8 var-9 var-10 var-11))
    (setf iinfo var-12))))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (the fixnum lwkopt) 'double-float))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil info))))

```

dorml2 LAPACK

— dorml2.input —

```

)set break resume
)sys rm -f dorml2.output
)spool dorml2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dorml2.help —

```

=====
dorml2 examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DORML2 - the general real m by n matrix C with $Q * C$ if $SIDE = 'L'$ and $TRANS = 'N'$, or $Q' * C$ if $SIDE = 'L'$ and $TRANS = 'T'$, or $C * Q$ if $SIDE = 'R'$ and $TRANS = 'N'$, or $C * Q'$ if $SIDE = 'R'$ and $TRANS = 'T'$,

SYNOPSIS

```

SUBROUTINE DORML2( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
                  INFO )

```

CHARACTER SIDE, TRANS

INTEGER INFO, K, LDA, LDC, M, N

DOUBLE PRECISION A(LDA, *), C(LDC, *), TAU(*), WORK(*)

Purpose

```

=====

```

DORML2 overwrites the general real m by n matrix C with

$Q * C$ if $SIDE = 'L'$ and $TRANS = 'N'$, or

$Q' * C$ if $SIDE = 'L'$ and $TRANS = 'T'$, or

$C * Q$ if SIDE = 'R' and TRANS = 'N', or

$C * Q'$ if SIDE = 'R' and TRANS = 'T',

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

$$Q = H(k) \dots H(2) H(1)$$

as returned by DGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

=====

SIDE (input) CHARACTER*1
 = 'L': apply Q or Q' from the Left
 = 'R': apply Q or Q' from the Right

TRANS (input) CHARACTER*1
 = 'N': apply Q (No transpose)
 = 'T': apply Q' (Transpose)

M (input) INTEGER
 The number of rows of the matrix C . $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix C . $N \geq 0$.

K (input) INTEGER
 The number of elementary reflectors whose product defines the matrix Q .
 If SIDE = 'L', $M \geq K \geq 0$;
 if SIDE = 'R', $N \geq K \geq 0$.

A (input) DOUBLE PRECISION array, dimension
 (LDA,M) if SIDE = 'L',
 (LDA,N) if SIDE = 'R'
 The i -th row must contain the vector which defines the elementary reflector $H(i)$, for $i = 1, 2, \dots, k$, as returned by DGELQF in the first k rows of its array argument A .
 A is modified by the routine but restored on exit.

LDA (input) INTEGER
 The leading dimension of the array A . $LDA \geq \max(1, K)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
 TAU(i) must contain the scalar factor of the elementary reflector $H(i)$, as returned by DGELQF.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the m by n matrix C.
 On exit, C is overwritten by Q*C or Q'*C or C*Q' or C*Q.

LDC (input) INTEGER
 The leading dimension of the array C. LDC $\geq \max(1,M)$.

WORK (workspace) DOUBLE PRECISION array, dimension
 (N) if SIDE = 'L',
 (M) if SIDE = 'R'

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

— dorml2.f —

```

SUBROUTINE DORML2( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
$                WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    February 29, 1992
*
*    .. Scalar Arguments ..
*    CHARACTER          SIDE, TRANS
*    INTEGER            INFO, K, LDA, LDC, M, N
*
*    ..
*    .. Array Arguments ..
*    DOUBLE PRECISION   A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*
*    ..
*
*    =====
*
*    .. Parameters ..
*    DOUBLE PRECISION   ONE
*    PARAMETER          ( ONE = 1.0D+0 )
*
*    ..
*    .. Local Scalars ..
*    LOGICAL            LEFT, NOTRAN
*    INTEGER            I, I1, I2, I3, IC, JC, MI, NI, NQ
*    DOUBLE PRECISION   AII
*
*    ..
*    .. External Functions ..

```

```

        LOGICAL          LSAME
        EXTERNAL          LSAME
*
*      ..
*      .. External Subroutines ..
        EXTERNAL          DLARF, XERBLA
*
*      ..
*      .. Intrinsic Functions ..
        INTRINSIC          MAX
*
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
        INFO = 0
        LEFT = LSAME( SIDE, 'L' )
        NOTRAN = LSAME( TRANS, 'N' )
*
*      NQ is the order of Q
*
        IF( LEFT ) THEN
            NQ = M
        ELSE
            NQ = N
        END IF
        IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
            INFO = -1
        ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) ) THEN
            INFO = -2
        ELSE IF( M.LT.0 ) THEN
            INFO = -3
        ELSE IF( N.LT.0 ) THEN
            INFO = -4
        ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
            INFO = -5
        ELSE IF( LDA.LT.MAX( 1, K ) ) THEN
            INFO = -7
        ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
            INFO = -10
        END IF
        IF( INFO.NE.0 ) THEN
            CALL XERBLA( 'DORML2', -INFO )
            RETURN
        END IF
*
*      Quick return if possible
*
        IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 )
$      RETURN
*
        IF( ( LEFT .AND. NOTRAN ) .OR. ( .NOT.LEFT .AND. .NOT.NOTRAN ) )

```

```

$      THEN
      I1 = 1
      I2 = K
      I3 = 1
    ELSE
      I1 = K
      I2 = 1
      I3 = -1
    END IF
*
    IF( LEFT ) THEN
      NI = N
      JC = 1
    ELSE
      MI = M
      IC = 1
    END IF
*
    DO 10 I = I1, I2, I3
      IF( LEFT ) THEN
*
*        H(i) is applied to C(i:m,1:n)
*
*        MI = M - I + 1
*        IC = I
*      ELSE
*
*        H(i) is applied to C(1:m,i:n)
*
*        NI = N - I + 1
*        JC = I
*      END IF
*
*      Apply H(i)
*
      AII = A( I, I )
      A( I, I ) = ONE
      CALL DLARF( SIDE, MI, NI, A( I, I ), LDA, TAU( I ),
$              C( IC, JC ), LDC, WORK )
      A( I, I ) = AII
10 CONTINUE
    RETURN
*
*    End of DORML2
*
    END

```

— LAPACK dorml2 —

```

(let* ((one 1.0))
  (declare (type (double-float 1.0 1.0) one))
  (defun dorml2 (side trans m n k a lda tau c ldc work info)
    (declare (type (simple-array double-float (*)) work c tau a)
      (type fixnum info ldc lda k n m)
      (type character trans side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((a1i 0.0) (i 0) (i1 0) (i2 0) (i3 0) (ic 0) (jc 0) (mi 0) (ni 0)
        (nq 0) (left nil) (notran nil))
        (declare (type (double-float) a1i)
          (type fixnum i i1 i2 i3 ic jc mi ni nq)
          (type (member t nil) left notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (cond
          (left
            (setf nq m))
          (t
            (setf nq n)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
            (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
            (setf info -2))
          ((< m 0)
            (setf info -3))
          ((< n 0)
            (setf info -4))
          ((or (< k 0) (> k nq))
            (setf info -5))
          ((< lda (max (the fixnum 1) (the fixnum k)))
            (setf info -7))
          ((< ldc (max (the fixnum 1) (the fixnum m)))
            (setf info -10)))
        (cond
          ((/= info 0)
            (error
              " ** On entry to ~a parameter number ~a had an illegal value~%"
              "DORML2" (f2cl-lib:int-sub info))
            (go end_label)))

```

```

(if (or (= m 0) (= n 0) (= k 0)) (go end_label))
(cond
  ((or (and left notran) (and (not left) (not notran)))
    (setf i1 1)
    (setf i2 k)
    (setf i3 1))
  (t
    (setf i1 k)
    (setf i2 1)
    (setf i3 -1)))
(cond
  (left
    (setf ni n)
    (setf jc 1))
  (t
    (setf mi m)
    (setf ic 1)))
(f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
  (> i i2) nil)
(tagbody
  (cond
    (left
      (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
      (setf ic i))
    (t
      (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
      (setf jc i)))
  (setf aii
    (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%))
  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
    one)
  (dlarf side mi ni
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
    (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
    (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *))) ldc
    work)
  (setf (f2cl-lib:fref a-%data% (i i) ((1 lda) (1 *)) a-%offset%)
    aii)))
end_label
(return (values nil nil nil nil nil nil nil nil nil nil info))))

```

dormlq LAPACK

— dormlq.input —

```

)set break resume
)sys rm -f dormlq.output
)spool dormlq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dormlq.help —

```

=====
dormlq examples
=====

```

```

=====
Man Page Details
=====

```

NAME

DORMLQ - the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R'
TRANS = 'N'

SYNOPSIS

SUBROUTINE DORMLQ(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
LWORK, INFO)

CHARACTER SIDE, TRANS

INTEGER INFO, K, LDA, LDC, LWORK, M, N

DOUBLE PRECISION A(LDA, *), C(LDC, *), TAU(*), WORK(
*)

Purpose

```

=====

```

DORMLQ overwrites the general real M-by-N matrix C with

| | | |
|--------------|------------|------------|
| | SIDE = 'L' | SIDE = 'R' |
| TRANS = 'N': | Q * C | C * Q |
| TRANS = 'T': | Q**T * C | C * Q**T |

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

$Q = H(k) \dots H(2) H(1)$

as returned by DGELQF. Q is of order M if $SIDE = 'L'$ and of order N if $SIDE = 'R'$.

Arguments =====

SIDE (input) CHARACTER*1
 = 'L': apply Q or Q^{**T} from the Left;
 = 'R': apply Q or Q^{**T} from the Right.

TRANS (input) CHARACTER*1
 = 'N': No transpose, apply Q ;
 = 'T': Transpose, apply Q^{**T} .

M (input) INTEGER
 The number of rows of the matrix C . $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix C . $N \geq 0$.

K (input) INTEGER
 The number of elementary reflectors whose product defines the matrix Q .
 If $SIDE = 'L'$, $M \geq K \geq 0$;
 if $SIDE = 'R'$, $N \geq K \geq 0$.

A (input) DOUBLE PRECISION array, dimension
 (LDA,M) if $SIDE = 'L'$,
 (LDA,N) if $SIDE = 'R'$
 The i -th row must contain the vector which defines the elementary reflector $H(i)$, for $i = 1, 2, \dots, k$, as returned by DGELQF in the first k rows of its array argument A .
 A is modified by the routine but restored on exit.

LDA (input) INTEGER
 The leading dimension of the array A . $LDA \geq \max(1, K)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
 TAU(i) must contain the scalar factor of the elementary reflector $H(i)$, as returned by DGELQF.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the M -by- N matrix C .
 On exit, C is overwritten by $Q \cdot C$ or $Q^{**T} \cdot C$ or $C \cdot Q^{**T}$ or $C \cdot Q$.

LDC (input) INTEGER
 The leading dimension of the array C . $LDC \geq \max(1, M)$.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If SIDE = 'L', LWORK $\geq \max(1, N)$;
if SIDE = 'R', LWORK $\geq \max(1, M)$.
For optimum performance LWORK $\geq N \cdot \text{NB}$ if SIDE = 'L', and
LWORK $\geq M \cdot \text{NB}$ if SIDE = 'R', where NB is the optimal
blocksize.

If LWORK = -1, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

— dormlq.f —

```

      SUBROUTINE DORMLQ( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
$                   WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
*
*  ..
*
*  .. Array Arguments ..
      DOUBLE PRECISION  A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
      INTEGER            NBMAX, LDT
      PARAMETER          ( NBMAX = 64, LDT = NBMAX+1 )
*
*  ..
*
*  .. Local Scalars ..
      LOGICAL            LEFT, LQUERY, NOTRAN

```

```

      CHARACTER          TRANST
      INTEGER            I, I1, I2, I3, IB, IC, IINFO, IWS, JC, LDWORK,
$      LWKOPT, MI, NB, NBMIN, NI, NQ, NW
*      ..
*      .. Local Arrays ..
      DOUBLE PRECISION   T( LDT, NBMAX )
*      ..
*      .. External Functions ..
      LOGICAL            LSAME
      INTEGER            ILAENV
      EXTERNAL           LSAME, ILAENV
*      ..
*      .. External Subroutines ..
      EXTERNAL           DLARFB, DLARFT, DORML2, XERBLA
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      LEFT = LSAME( SIDE, 'L' )
      NOTRAN = LSAME( TRANS, 'N' )
      LQUERY = ( LWORK.EQ.-1 )
*
*      NQ is the order of Q and NW is the minimum dimension of WORK
*
      IF( LEFT ) THEN
         NQ = M
         NW = N
      ELSE
         NQ = N
         NW = M
      END IF
      IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
         INFO = -1
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) ) THEN
         INFO = -2
      ELSE IF( M.LT.0 ) THEN
         INFO = -3
      ELSE IF( N.LT.0 ) THEN
         INFO = -4
      ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
         INFO = -5
      ELSE IF( LDA.LT.MAX( 1, K ) ) THEN
         INFO = -7
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
         INFO = -10

```

```

ELSE IF( LWORK.LT.MAX( 1, NW ) .AND. .NOT.LQUERY ) THEN
    INFO = -12
END IF
*
IF( INFO.EQ.0 ) THEN
*
*   Determine the block size.  NB may be at most NBMAX, where NBMAX
*   is used to define the local array T.
*
    NB = MIN( NBMAX, ILAENV( 1, 'DORMLQ', SIDE // TRANS, M, N, K,
$      -1 ) )
    LWKOPT = MAX( 1, NW )*NB
    WORK( 1 ) = LWKOPT
END IF
*
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DORMLQ', -INFO )
    RETURN
ELSE IF( LQUERY ) THEN
    RETURN
END IF
*
*   Quick return if possible
*
IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 ) THEN
    WORK( 1 ) = 1
    RETURN
END IF
*
NBMIN = 2
LDWORK = NW
IF( NB.GT.1 .AND. NB.LT.K ) THEN
    IWS = NW*NB
    IF( LWORK.LT.IWS ) THEN
        NB = LWORK / LDWORK
        NBMIN = MAX( 2, ILAENV( 2, 'DORMLQ', SIDE // TRANS, M, N, K,
$      -1 ) )
    END IF
ELSE
    IWS = NW
END IF
*
IF( NB.LT.NBMIN .OR. NB.GE.K ) THEN
*
*   Use unblocked code
*
    CALL DORML2( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
$      IINFO )
ELSE
*

```

```

*      Use blocked code
*
      IF( ( LEFT .AND. NOTRAN ) .OR.
$      ( .NOT.LEFT .AND. .NOT.NOTRAN ) ) THEN
          I1 = 1
          I2 = K
          I3 = NB
      ELSE
          I1 = ( ( K-1 ) / NB ) * NB + 1
          I2 = 1
          I3 = -NB
      END IF
*
      IF( LEFT ) THEN
          NI = N
          JC = 1
      ELSE
          MI = M
          IC = 1
      END IF
*
      IF( NOTRAN ) THEN
          TRANST = 'T'
      ELSE
          TRANST = 'N'
      END IF
*
      DO 10 I = I1, I2, I3
          IB = MIN( NB, K-I+1 )
*
*      Form the triangular factor of the block reflector
*      H = H(i) H(i+1) . . . H(i+ib-1)
*
          CALL DLARFT( 'Forward', 'Rowwise', NQ-I+1, IB, A( I, I ),
$              LDA, TAU( I ), T, LDT )
          IF( LEFT ) THEN
*
*              H or H' is applied to C(i:m,1:n)
*
              MI = M - I + 1
              IC = I
          ELSE
*
*              H or H' is applied to C(1:m,i:n)
*
              NI = N - I + 1
              JC = I
          END IF
*
*      Apply H or H'

```



```

*
      CALL DLARFB( SIDE, TRANST, 'Forward', 'Rowwise', MI, NI, IB,
$           A( I, I ), LDA, T, LDT, C( IC, JC ), LDC, WORK,
$           LDWORK )
10    CONTINUE
      END IF
      WORK( 1 ) = LWKOPT
      RETURN
*
*    End of DORMLQ
*
      END

```

— LAPACK dormlq —

```

(let* ((nbmax 64) (ldt (+ nbmax 1)))
  (declare (type (fixnum 64 64) nbmax)
            (type fixnum ldt))
  (defun dormlq (side trans m n k a lda tau c ldc work lwork info)
    (declare (type (simple-array double-float (*)) work c tau a)
              (type fixnum info lwork ldc lda k n m)
              (type character trans side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (i1 0) (i2 0) (i3 0) (ib 0) (ic 0) (iinfo 0) (iws 0) (jc 0)
              (ldwork 0) (lwkopt 0) (mi 0) (nb 0) (nbmin 0) (ni 0) (nq 0) (nw 0)
              (transt
                (make-array '(1) :element-type 'character :initial-element #\ )
                (left nil) (lquery nil) (notran nil)
                (t$
                  (make-array (the fixnum (reduce #'* (list ldt nbmax)))
                              :element-type 'double-float)))
              (declare (type (simple-array double-float (*)) t$)
                        (type fixnum i i1 i2 i3 ib ic iinfo iws jc ldwork
                                  lwkopt mi nb nbmin ni nq nw)
                        (type (simple-array character (1)) transt)
                        (type (member t nil) left lquery notran))
              (setf info 0)
              (setf left (char-equal side #\L))
              (setf notran (char-equal trans #\N))
              (setf lquery (coerce (= lwork -1) '(member t nil))))

```

```

(cond
  (left
    (setf nq m)
    (setf nw n))
  (t
    (setf nq n)
    (setf nw m)))
(cond
  ((and (not left) (not (char-equal side #\R)))
    (setf info -1))
  ((and (not notran) (not (char-equal trans #\T)))
    (setf info -2))
  ((< m 0)
    (setf info -3))
  ((< n 0)
    (setf info -4))
  ((or (< k 0) (> k nq))
    (setf info -5))
  ((< lda (max (the fixnum 1) (the fixnum k)))
    (setf info -7))
  ((< ldc (max (the fixnum 1) (the fixnum m)))
    (setf info -10))
  ((and
    (< lwork
      (max (the fixnum 1) (the fixnum nw)))
    (not lquery))
    (setf info -12)))
(cond
  ((= info 0)
    (setf nb
      (min (the fixnum nbmax)
        (the fixnum
          (ilaenv 1 "DORMLQ" (f2cl-lib:f2cl-// side trans) m
            n k -1)))))
  (setf lwkopt
    (f2cl-lib:int-mul
      (max (the fixnum 1) (the fixnum nw))
      nb))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (coerce (the fixnum lwkopt) 'double-float)))
(cond
  ((/= info 0)
    (error
      " ** On entry to ~a parameter number ~a had an illegal value~%"
      "DORMLQ" (f2cl-lib:int-sub info))
    (go end_label))
  (lquery
    (go end_label)))
(cond
  ((or (= m 0) (= n 0) (= k 0))

```

```

      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
            (coerce (the fixnum 1) 'double-float))
      (go end_label)))
(setf nbmin 2)
(setf ldwork nw)
(cond
 ((and (> nb 1) (< nb k))
  (setf iws (f2cl-lib:int-mul nw nb))
  (cond
   ((< lwork iws)
    (setf nb (the fixnum (truncate lwork ldwork)))
    (setf nbmin
         (max (the fixnum 2)
              (the fixnum
               (ilaenv 2 "DORMLQ"
                       (f2cl-lib:f2cl-// side trans) m n k -1))))))
   (t
    (setf iws nw))))
(cond
 ((or (< nb nbmin) (>= nb k))
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
    var-10 var-11)
   (dorml2 side trans m n k a lda tau c ldc work iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                   var-8 var-9 var-10))
  (setf iinfo var-11)))
 (t
  (cond
   ((or (and left notran) (and (not left) (not notran)))
    (setf i1 1)
    (setf i2 k)
    (setf i3 nb))
    (t
     (setf i1
          (+ (* (the fixnum (truncate (- k 1) nb)) nb)
             1))
      (setf i2 1)
      (setf i3 (f2cl-lib:int-sub nb))))
   (cond
    (left
     (setf ni n)
     (setf jc 1))
    (t
     (setf mi m)
     (setf ic 1)))
   (cond
    (notran
     (f2cl-lib:f2cl-set-string transt "T" (string 1)))
    (t

```

```

(f2cl-lib:f2cl-set-string transt "N" (string 1)))
(f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
  (> i i2) nil)
(tagbody
  (setf ib
    (min (the fixnum nb)
      (the fixnum
        (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
  (dlarft "Forward" "Rowwise"
    (f2cl-lib:int-add (f2cl-lib:int-sub nq i) 1) ib
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
    (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt)
  (cond
    (left
      (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
      (setf ic i))
    (t
      (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
      (setf jc i)))
  (dlarfb side transt "Forward" "Rowwise" mi ni ib
    (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
    t$ ldt
    (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *)))
    ldc work ldwork))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce (the fixnum lwkopt) 'double-float))
end_label
(return
  (values nil nil nil nil nil nil nil nil nil nil nil nil info))))

```

dormqr LAPACK

— dormqr.input —

```

)set break resume
)sys rm -f dormqr.output
)spool dormqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dormqr.help —

=====

dormqr examples

=====

=====

Man Page Details

=====

NAME

DORMQR - the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R'
TRANS = 'N'

SYNOPSIS

SUBROUTINE DORMQR(SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
LWORK, INFO)

CHARACTER SIDE, TRANS

INTEGER INFO, K, LDA, LDC, LWORK, M, N

DOUBLE PRECISION A(LDA, *), C(LDC, *), TAU(*), WORK(
*)

Purpose

=====

DORMQR overwrites the general real M-by-N matrix C with

| | | |
|--------------|------------|------------|
| | SIDE = 'L' | SIDE = 'R' |
| TRANS = 'N': | Q * C | C * Q |
| TRANS = 'T': | Q**T * C | C * Q**T |

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Arguments

=====

SIDE (input) CHARACTER*1
= 'L': apply Q or Q**T from the Left;

= 'R': apply Q or Q^{**T} from the Right.

TRANS (input) CHARACTER*1
 = 'N': No transpose, apply Q;
 = 'T': Transpose, apply Q^{**T} .

M (input) INTEGER
 The number of rows of the matrix C. $M \geq 0$.

N (input) INTEGER
 The number of columns of the matrix C. $N \geq 0$.

K (input) INTEGER
 The number of elementary reflectors whose product defines the matrix Q.
 If SIDE = 'L', $M \geq K \geq 0$;
 if SIDE = 'R', $N \geq K \geq 0$.

A (input) DOUBLE PRECISION array, dimension (LDA,K)
 The i-th column must contain the vector which defines the elementary reflector $H(i)$, for $i = 1, 2, \dots, k$, as returned by DGEQRF in the first k columns of its array argument A.
 A is modified by the routine but restored on exit.

LDA (input) INTEGER
 The leading dimension of the array A.
 If SIDE = 'L', $LDA \geq \max(1, M)$;
 if SIDE = 'R', $LDA \geq \max(1, N)$.

TAU (input) DOUBLE PRECISION array, dimension (K)
 TAU(i) must contain the scalar factor of the elementary reflector $H(i)$, as returned by DGEQRF.

C (input/output) DOUBLE PRECISION array, dimension (LDC,N)
 On entry, the M-by-N matrix C.
 On exit, C is overwritten by Q^*C or $Q^{**T}C$ or CQ^{**T} or CQ .

LDC (input) INTEGER
 The leading dimension of the array C. $LDC \geq \max(1, M)$.

WORK (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
 On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
 The dimension of the array WORK.
 If SIDE = 'L', $LWORK \geq \max(1, N)$;
 if SIDE = 'R', $LWORK \geq \max(1, M)$.
 For optimum performance $LWORK \geq N \cdot NB$ if SIDE = 'L', and
 $LWORK \geq M \cdot NB$ if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

— dormqr.f —

```

      SUBROUTINE DORMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
$                      WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
*  CHARACTER            SIDE, TRANS
*  INTEGER              INFO, K, LDA, LDC, LWORK, M, N
*
*  ..
*  .. Array Arguments ..
*  DOUBLE PRECISION    A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*
*  ..
*  =====
*
*  .. Parameters ..
*  INTEGER              NBMAX, LDT
*  PARAMETER            ( NBMAX = 64, LDT = NBMAX+1 )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL              LEFT, LQUERY, NOTRAN
*  INTEGER              I, I1, I2, I3, IB, IC, IINFO, IWS, JC, LDWORK,
$                      LWKOPT, MI, NB, NBMIN, NI, NQ, NW
*
*  ..
*  .. Local Arrays ..
*  DOUBLE PRECISION    T( LDT, NBMAX )
*
*  ..
*  .. External Functions ..
*  LOGICAL              LSAME
*  INTEGER              ILAENV
*  EXTERNAL             LSAME, ILAENV

```

```

*      ..
*      .. External Subroutines ..
EXTERNAL          DLARFB, DLARFT, DORM2R, XERBLA
*      ..
*      .. Intrinsic Functions ..
INTRINSIC          MAX, MIN
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
      INFO = 0
      LEFT = LSAME( SIDE, 'L' )
      NOTRAN = LSAME( TRANS, 'N' )
      LQUERY = ( LWORK.EQ.-1 )
*
*      NQ is the order of Q and NW is the minimum dimension of WORK
*
      IF( LEFT ) THEN
         NQ = M
         NW = N
      ELSE
         NQ = N
         NW = M
      END IF
      IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
         INFO = -1
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) ) THEN
         INFO = -2
      ELSE IF( M.LT.0 ) THEN
         INFO = -3
      ELSE IF( N.LT.0 ) THEN
         INFO = -4
      ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
         INFO = -5
      ELSE IF( LDA.LT.MAX( 1, NQ ) ) THEN
         INFO = -7
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
         INFO = -10
      ELSE IF( LWORK.LT.MAX( 1, NW ) .AND. .NOT.LQUERY ) THEN
         INFO = -12
      END IF
*
      IF( INFO.EQ.0 ) THEN
*
*         Determine the block size.  NB may be at most NBMAX, where NBMAX
*         is used to define the local array T.
*
         NB = MIN( NBMAX, ILAENV( 1, 'DORMQR', SIDE // TRANS, M, N, K,
$           -1 ) )

```



```

        LWKOPT = MAX( 1, NW ) * NB
        WORK( 1 ) = LWKOPT
    END IF

*
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'DORMQR', -INFO )
        RETURN
    ELSE IF( LQUERY ) THEN
        RETURN
    END IF

*
*   Quick return if possible
*
    IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 ) THEN
        WORK( 1 ) = 1
        RETURN
    END IF

*
    NBMIN = 2
    LDWORK = NW
    IF( NB.GT.1 .AND. NB.LT.K ) THEN
        IWS = NW * NB
        IF( LWORK.LT.IWS ) THEN
            NB = LWORK / LDWORK
            NBMIN = MAX( 2, ILAENV( 2, 'DORMQR', SIDE // TRANS, M, N, K,
$              -1 ) )
        END IF
    ELSE
        IWS = NW
    END IF

*
    IF( NB.LT.NBMIN .OR. NB.GE.K ) THEN

*
*       Use unblocked code
*
        CALL DORM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
$              IINFO )
    ELSE

*
*       Use blocked code
*
    IF( ( LEFT .AND. .NOT.NOTRAN ) .OR.
$      ( .NOT.LEFT .AND. NOTRAN ) ) THEN
        I1 = 1
        I2 = K
        I3 = NB
    ELSE
        I1 = ( ( K-1 ) / NB ) * NB + 1
        I2 = 1
        I3 = -NB
    
```

```

      END IF
*
      IF( LEFT ) THEN
        NI = N
        JC = 1
      ELSE
        MI = M
        IC = 1
      END IF
*
      DO 10 I = I1, I2, I3
        IB = MIN( NB, K-I+1 )
*
*        Form the triangular factor of the block reflector
*        H = H(i) H(i+1) . . . H(i+ib-1)
*
        CALL DLARFT( 'Forward', 'Columnwise', NQ-I+1, IB, A( I, I ),
$              LDA, TAU( I ), T, LDT )
        IF( LEFT ) THEN
*
*          H or H' is applied to C(i:m,i:n)
*
*          MI = M - I + 1
*          IC = I
*        ELSE
*
*          H or H' is applied to C(1:m,i:n)
*
*          NI = N - I + 1
*          JC = I
*        END IF
*
*        Apply H or H'
*
        CALL DLARFB( SIDE, TRANS, 'Forward', 'Columnwise', MI, NI,
$              IB, A( I, I ), LDA, T, LDT, C( IC, JC ), LDC,
$              WORK, LDWORK )
10    CONTINUE
      END IF
      WORK( 1 ) = LWKOPT
      RETURN
*
*    End of DORMQR
*
      END

```

— LAPACK dormqr —

```

(let* ((nbmax 64) (ldt (+ nbmax 1)))
  (declare (type (fixnum 64 64) nbmax)
    (type fixnum ldt))
  (defun dormqr (side trans m n k a lda tau c ldc work lwork info)
    (declare (type (simple-array double-float (*)) work c tau a)
      (type fixnum info lwork ldc lda k n m)
      (type character trans side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (a double-float a-%data% a-%offset%)
       (tau double-float tau-%data% tau-%offset%)
       (c double-float c-%data% c-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((i 0) (i1 0) (i2 0) (i3 0) (ib 0) (ic 0) (iinfo 0) (iws 0) (jc 0)
        (ldwork 0) (lwkopt 0) (mi 0) (nb 0) (nbmin 0) (ni 0) (nq 0) (nw 0)
        (left nil) (lquery nil) (notran nil)
        (t$
          (make-array (the fixnum (reduce #'* (list ldt nbmax)))
            :element-type 'double-float)))
        (declare (type (simple-array double-float (*)) t$)
          (type fixnum i i1 i2 i3 ib ic iinfo iws jc ldwork
            lwkopt mi nb nbmin ni nq nw)
          (type (member t nil) left lquery notran))
        (setf info 0)
        (setf left (char-equal side #\L))
        (setf notran (char-equal trans #\N))
        (setf lquery (coerce (= lwork -1) '(member t nil)))
        (cond
          (left
            (setf nq m)
            (setf nw n))
          (t
            (setf nq n)
            (setf nw m)))
        (cond
          ((and (not left) (not (char-equal side #\R)))
            (setf info -1))
          ((and (not notran) (not (char-equal trans #\T)))
            (setf info -2))
          ((< m 0)
            (setf info -3))
          ((< n 0)
            (setf info -4))
          ((or (< k 0) (> k nq))
            (setf info -5))
          ((< lda (max (the fixnum 1) (the fixnum nq)))
            (setf info -7))

```

```

((< ldc (max (the fixnum 1) (the fixnum m)))
 (setf info -10))
(and
 (< lwork
  (max (the fixnum 1) (the fixnum nw)))
 (not lquery))
 (setf info -12))
(cond
 ((= info 0)
  (setf nb
   (min (the fixnum nbmax)
        (the fixnum
         (ilaenv 1 "DORMQR" (f2cl-lib:f2cl-// side trans) m
                   n k -1)))))
  (setf lwkopt
   (f2cl-lib:int-mul
    (max (the fixnum 1) (the fixnum nw))
    nb))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
   (coerce (the fixnum lwkopt) 'double-float)))
(cond
 (/= info 0)
 (error
  " ** On entry to ~a parameter number ~a had an illegal value~%"
  "DORMQR" (f2cl-lib:int-sub info))
 (go end_label))
(lquery
 (go end_label)))
(cond
 ((or (= m 0) (= n 0) (= k 0))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
   (coerce (the fixnum 1) 'double-float))
  (go end_label)))
(setf nbmin 2)
(setf ldwork nw)
(cond
 ((and (> nb 1) (< nb k))
  (setf iws (f2cl-lib:int-mul nw nb))
  (cond
   ((< lwork iws)
    (setf nb (the fixnum (truncate lwork ldwork)))
    (setf nbmin
     (max (the fixnum 2)
          (the fixnum
           (ilaenv 2 "DORMQR"
                    (f2cl-lib:f2cl-// side trans) m n k -1))))))
   (t
    (setf iws nw))))
(cond
 ((or (< nb nbmin) (>= nb k))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11)
  (dorm2r side trans m n k a lda tau c ldc work iinfo)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
                  var-8 var-9 var-10))
  (setf iinfo var-11)))
(t
 (cond
  ((or (and left (not notran)) (and (not left) notran))
   (setf i1 1)
   (setf i2 k)
   (setf i3 nb))
  (t
   (setf i1
         (+ (* (the fixnum (truncate (- k 1) nb)) nb)
            1))
   (setf i2 1)
   (setf i3 (f2cl-lib:int-sub nb))))
 (cond
  (left
   (setf ni n)
   (setf jc 1))
  (t
   (setf mi m)
   (setf ic 1)))
 (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
                (> i i2) nil)
 (tagbody
  (setf ib
        (min (the fixnum nb)
              (the fixnum
                (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1))))
  (dlarft "Forward" "Columnwise"
          (f2cl-lib:int-add (f2cl-lib:int-sub nq i) 1) ib
          (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
          (f2cl-lib:array-slice tau double-float (i) ((1 *))) t$ ldt)
  (cond
   (left
    (setf mi (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1))
    (setf ic i))
   (t
    (setf ni (f2cl-lib:int-add (f2cl-lib:int-sub n i) 1))
    (setf jc i)))
  (dlarfb side trans "Forward" "Columnwise" mi ni ib
          (f2cl-lib:array-slice a double-float (i i) ((1 lda) (1 *))) lda
          t$ ldt
          (f2cl-lib:array-slice c double-float (ic jc) ((1 ldc) (1 *)))
          ldc work ldwork))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)

```

```

                                (coerce (the fixnum lwkopt) 'double-float))
end_label
  (return
    (values nil nil nil nil nil nil nil nil nil nil nil nil info))))))

```

dtrevc LAPACK

— dtrevc.input —

```

)set break resume
)sys rm -f dtrevc.output
)spool dtrevc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrevc.help —

```

=====
dtrevc examples
=====

=====
Man Page Details
=====

```

NAME

DTREVC - some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T

SYNOPSIS

```

SUBROUTINE DTREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR,
                  MM, M, WORK, INFO )

```

CHARACTER HOWMNY, SIDE

INTEGER INFO, LDT, LDVL, LDVR, M, MM, N

```

LOGICAL      SELECT( * )

DOUBLE       PRECISION T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
              WORK( * )

```

Purpose
=====

DTREVC computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

$$T*x = w*x, \quad y'*T = w*y'$$

where y' denotes the conjugate transpose of the vector y.

If all eigenvectors are requested, the routine may either return the matrices X and/or Y of right or left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input orthogonal matrix. If T was obtained from the real-Schur factorization of an original matrix $A = Q*T*Q'$, then Q*X and Q*Y are the matrices of right or left eigenvectors of A.

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign. Corresponding to each 2-by-2 diagonal block is a complex conjugate pair of eigenvalues and eigenvectors; only one eigenvector of the pair is computed, namely the one corresponding to the eigenvalue with positive imaginary part.

Arguments
=====

```

SIDE      (input) CHARACTER*1
          = 'R':  compute right eigenvectors only;
          = 'L':  compute left eigenvectors only;
          = 'B':  compute both right and left eigenvectors.

HOWMNY    (input) CHARACTER*1
          = 'A':  compute all right and/or left eigenvectors;
          = 'B':  compute all right and/or left eigenvectors,
                  and backtransform them using the input matrices
                  supplied in VR and/or VL;
          = 'S':  compute selected right and/or left eigenvectors,
                  specified by the logical array SELECT.

SELECT    (input/output) LOGICAL array, dimension (N)

```

If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed.
 If HOWMNY = 'A' or 'B', SELECT is not referenced.
 To select the real eigenvector corresponding to a real eigenvalue $w(j)$, SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex conjugate pair $w(j)$ and $w(j+1)$, either SELECT(j) or SELECT(j+1) must be set to .TRUE.; then on exit SELECT(j) is .TRUE. and SELECT(j+1) is .FALSE..

- N (input) INTEGER
 The order of the matrix T. $N \geq 0$.
- T (input) DOUBLE PRECISION array, dimension (LDT,N)
 The upper quasi-triangular matrix T in Schur canonical form.
- LDT (input) INTEGER
 The leading dimension of the array T. $LDT \geq \max(1,N)$.
- VL (input/output) DOUBLE PRECISION array, dimension (LDVL,MM)
 On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR).
 On exit, if SIDE = 'L' or 'B', VL contains:
 if HOWMNY = 'A', the matrix Y of left eigenvectors of T;
 VL has the same quasi-lower triangular form as T'. If $T(i,i)$ is a real eigenvalue, then the i-th column VL(i) of VL is its corresponding eigenvector. If $T(i:i+1,i:i+1)$ is a 2-by-2 block whose eigenvalues are complex-conjugate eigenvalues of T, then $VL(i)+\sqrt{-1}*VL(i+1)$ is the complex eigenvector corresponding to the eigenvalue with positive real part.
 if HOWMNY = 'B', the matrix $Q*Y$;
 if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues.
 A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part.
 If SIDE = 'R', VL is not referenced.
- LDVL (input) INTEGER
 The leading dimension of the array VL. $LDVL \geq \max(1,N)$ if SIDE = 'L' or 'B'; $LDVL \geq 1$ otherwise.
- VR (input/output) DOUBLE PRECISION array, dimension (LDVR,MM)
 On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must

contain an N-by-N matrix Q (usually the orthogonal matrix Q of Schur vectors returned by DHSEQR).

On exit, if SIDE = 'R' or 'B', VR contains:

if HOWMNY = 'A', the matrix X of right eigenvectors of T; VR has the same quasi-upper triangular form as T. If T(i,i) is a real eigenvalue, then the i-th column VR(i) of VR is its corresponding eigenvector. If T(i:i+1,i:i+1) is a 2-by-2 block whose eigenvalues are complex-conjugate eigenvalues of T, then VR(i)+sqrt(-1)*VR(i+1) is the complex eigenvector corresponding to the eigenvalue with positive real part.

if HOWMNY = 'B', the matrix Q*X;

if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues.

A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

If SIDE = 'L', VR is not referenced.

LDVR (input) INTEGER
The leading dimension of the array VR. LDVR \geq max(1,N) if SIDE = 'R' or 'B'; LDVR \geq 1 otherwise.

MM (input) INTEGER
The number of columns in the arrays VL and/or VR. MM \geq M.

M (output) INTEGER
The number of columns in the arrays VL and/or VR actually used to store the eigenvectors.
If HOWMNY = 'A' or 'B', M is set to N.
Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.

WORK (workspace) DOUBLE PRECISION array, dimension (3*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

Further Details

=====

The algorithm used in this program is basically backward (forward) substitution, with scaling to make the the code robust against possible overflow.

Each eigenvector is normalized so that the element of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be $|x| + |y|$.

— dtrevc.f —

```

SUBROUTINE DTREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
$                LDVR, MM, M, WORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
*  CHARACTER          HOWMNY, SIDE
*  INTEGER            INFO, LDT, LDVL, LDVR, M, MM, N
*
*  ..
*  .. Array Arguments ..
*  LOGICAL            SELECT( * )
*  DOUBLE PRECISION   T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
$                   WORK( * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
*  DOUBLE PRECISION   ZERO, ONE
*  PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*
*  ..
*  .. Local Scalars ..
*  LOGICAL            ALLV, BOTHV, LEFTV, OVER, PAIR, RIGHTV, SOMEV
*  INTEGER            I, IERR, II, IP, IS, J, J1, J2, JNXT, K, KI, N2
*  DOUBLE PRECISION   BETA, BIGNUM, EMAX, OVFL, REC, REMAX, SCALE,
$                   SMIN, SMLNUM, ULP, UNFL, VCRIT, VMAX, WI, WR,
$                   XNORM
*
*  ..
*  .. External Functions ..
*  LOGICAL            LSAME
*  INTEGER            IDAMAX
*  DOUBLE PRECISION   DDOT, DLAMCH
*  EXTERNAL           LSAME, IDAMAX, DDOT, DLAMCH
*
*  ..
*  .. External Subroutines ..
*  EXTERNAL           DAXPY, DCOPY, DGEMV, DLALN2, DSCAL, XERBLA
*
*  ..

```

```

*      .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, SQRT
*
*      ..
*      .. Local Arrays ..
      DOUBLE PRECISION  X( 2, 2 )
*
*      ..
*      .. Executable Statements ..
*
*      Decode and test the input parameters
*
      BOTHV = LSAME( SIDE, 'B' )
      RIGHTV = LSAME( SIDE, 'R' ) .OR. BOTHV
      LEFTV = LSAME( SIDE, 'L' ) .OR. BOTHV
*
      ALLV = LSAME( HOWMNY, 'A' )
      OVER = LSAME( HOWMNY, 'B' )
      SOMEV = LSAME( HOWMNY, 'S' )
*
      INFO = 0
      IF( .NOT.RIGHTV .AND. .NOT.LEFTV ) THEN
         INFO = -1
      ELSE IF( .NOT.ALLV .AND. .NOT.OVER .AND. .NOT.SOMEV ) THEN
         INFO = -2
      ELSE IF( N.LT.0 ) THEN
         INFO = -4
      ELSE IF( LDT.LT.MAX( 1, N ) ) THEN
         INFO = -6
      ELSE IF( LDVL.LT.1 .OR. ( LEFTV .AND. LDVL.LT.N ) ) THEN
         INFO = -8
      ELSE IF( LDVR.LT.1 .OR. ( RIGHTV .AND. LDVR.LT.N ) ) THEN
         INFO = -10
      ELSE
*
*      Set M to the number of columns required to store the selected
*      eigenvectors, standardize the array SELECT if necessary, and
*      test MM.
*
      IF( SOMEV ) THEN
         M = 0
         PAIR = .FALSE.
         DO 10 J = 1, N
            IF( PAIR ) THEN
               PAIR = .FALSE.
               SELECT( J ) = .FALSE.
            ELSE
               IF( J.LT.N ) THEN
                  IF( T( J+1, J ).EQ.ZERO ) THEN
                     IF( SELECT( J ) )
                        $          M = M + 1
                  ELSE

```

```

        PAIR = .TRUE.
        IF( SELECT( J ) .OR. SELECT( J+1 ) ) THEN
            SELECT( J ) = .TRUE.
            M = M + 2
        END IF
    END IF
ELSE
    IF( SELECT( N ) )
$        M = M + 1
    END IF
END IF
10    CONTINUE
    ELSE
        M = N
    END IF
*
    IF( MM.LT.M ) THEN
        INFO = -11
    END IF
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'DTREVC', -INFO )
    RETURN
END IF
*
*    Quick return if possible.
*
    IF( N.EQ.0 )
$    RETURN
*
*    Set the constants to control overflow.
*
    UNFL = DLAMCH( 'Safe minimum' )
    OVFL = ONE / UNFL
    CALL DLABAD( UNFL, OVFL )
    ULP = DLAMCH( 'Precision' )
    SMLNUM = UNFL*( N / ULP )
    BIGNUM = ( ONE-ULP ) / SMLNUM
*
*    Compute 1-norm of each column of strictly upper triangular
*    part of T to control overflow in triangular solver.
*
    WORK( 1 ) = ZERO
    DO 30 J = 2, N
        WORK( J ) = ZERO
        DO 20 I = 1, J - 1
            WORK( J ) = WORK( J ) + ABS( T( I, J ) )
20        CONTINUE
30    CONTINUE
*

```

```

*      Index IP is used to specify the real or complex eigenvalue:
*      IP = 0, real eigenvalue,
*           1, first of conjugate complex pair: (wr,wi)
*          -1, second of conjugate complex pair: (wr,wi)
*
*      N2 = 2*N
*
*      IF( RIGHTV ) THEN
*
*          Compute right eigenvectors.
*
*          IP = 0
*          IS = M
*          DO 140 KI = N, 1, -1
*
*              IF( IP.EQ.1 )
*  $              GO TO 130
*              IF( KI.EQ.1 )
*  $              GO TO 40
*              IF( T( KI, KI-1 ).EQ.ZERO )
*  $              GO TO 40
*              IP = -1
*
* 40          CONTINUE
*              IF( SOMEV ) THEN
*                  IF( IP.EQ.0 ) THEN
*                      IF( .NOT.SELECT( KI ) )
*  $                      GO TO 130
*                  ELSE
*                      IF( .NOT.SELECT( KI-1 ) )
*  $                      GO TO 130
*                  END IF
*              END IF
*
*          Compute the KI-th eigenvalue (WR,WI).
*
*          WR = T( KI, KI )
*          WI = ZERO
*          IF( IP.NE.0 )
*  $          WI = SQRT( ABS( T( KI, KI-1 ) ) ) *
*  $          SQRT( ABS( T( KI-1, KI ) ) )
*          SMIN = MAX( ULP*( ABS( WR )+ABS( WI ) ), SMLNUM )
*
*          IF( IP.EQ.0 ) THEN
*
*              Real right eigenvector
*
*              WORK( KI+N ) = ONE
*
*              Form right-hand side

```

```

*
DO 50 K = 1, KI - 1
  WORK( K+N ) = -T( K, KI )
50 CONTINUE
*
* Solve the upper quasi-triangular system:
*   (T(1:KI-1,1:KI-1) - WR)*X = SCALE*WORK.
*
JNXT = KI - 1
DO 60 J = KI - 1, 1, -1
  IF( J.GT.JNXT )
    $ GO TO 60
    J1 = J
    J2 = J
    JNXT = J - 1
    IF( J.GT.1 ) THEN
      IF( T( J, J-1 ).NE.ZERO ) THEN
        J1 = J - 1
        JNXT = J - 2
      END IF
    END IF
  END IF

  IF( J1.EQ.J2 ) THEN

    1-by-1 diagonal block

    CALL DLALN2( .FALSE., 1, 1, SMIN, ONE, T( J, J ),
    $             LDT, ONE, ONE, WORK( J+N ), N, WR,
    $             ZERO, X, 2, SCALE, XNORM, IERR )

    *
    * Scale X(1,1) to avoid overflow when updating
    * the right-hand side.
    *
    IF( XNORM.GT.ONE ) THEN
      IF( WORK( J ).GT.BIGNUM / XNORM ) THEN
        X( 1, 1 ) = X( 1, 1 ) / XNORM
        SCALE = SCALE / XNORM
      END IF
    END IF

    *
    * Scale if necessary
    *
    IF( SCALE.NE.ONE )
      $ CALL DSCAL( KI, SCALE, WORK( 1+N ), 1 )
      WORK( J+N ) = X( 1, 1 )

    *
    * Update right-hand side
    *
    CALL DAXPY( J-1, -X( 1, 1 ), T( 1, J ), 1,
    $           WORK( 1+N ), 1 )

```

```

*
*           ELSE
*
*           2-by-2 diagonal block
*
*           CALL DLALN2( .FALSE., 2, 1, SMIN, ONE,
$              T( J-1, J-1 ), LDT, ONE, ONE,
$              WORK( J-1+N ), N, WR, ZERO, X, 2,
$              SCALE, XNORM, IERR )
*
*           Scale X(1,1) and X(2,1) to avoid overflow when
*           updating the right-hand side.
*
*           IF( XNORM.GT.ONE ) THEN
*             BETA = MAX( WORK( J-1 ), WORK( J ) )
*             IF( BETA.GT.BIGNUM / XNORM ) THEN
*               X( 1, 1 ) = X( 1, 1 ) / XNORM
*               X( 2, 1 ) = X( 2, 1 ) / XNORM
*               SCALE = SCALE / XNORM
*             END IF
*           END IF
*
*           Scale if necessary
*
*           IF( SCALE.NE.ONE )
$             CALL DSCAL( KI, SCALE, WORK( 1+N ), 1 )
$             WORK( J-1+N ) = X( 1, 1 )
$             WORK( J+N ) = X( 2, 1 )
*
*           Update right-hand side
*
*           CALL DAXPY( J-2, -X( 1, 1 ), T( 1, J-1 ), 1,
$             WORK( 1+N ), 1 )
$           CALL DAXPY( J-2, -X( 2, 1 ), T( 1, J ), 1,
$             WORK( 1+N ), 1 )
*           END IF
60      CONTINUE
*
*           Copy the vector x or Q*x to VR and normalize.
*
*           IF( .NOT.OVER ) THEN
*             CALL DCOPY( KI, WORK( 1+N ), 1, VR( 1, IS ), 1 )
*
*             II = IDAMAX( KI, VR( 1, IS ), 1 )
*             REMAX = ONE / ABS( VR( II, IS ) )
*             CALL DSCAL( KI, REMAX, VR( 1, IS ), 1 )
*
*             DO 70 K = KI + 1, N
*               VR( K, IS ) = ZERO
70      CONTINUE

```

```

ELSE
  IF( KI.GT.1 )
    $      CALL DGEMV( 'N', N, KI-1, ONE, VR, LDVR,
    $      WORK( 1+N ), 1, WORK( KI+N ),
    $      VR( 1, KI ), 1 )
  *
    II = IDAMAX( N, VR( 1, KI ), 1 )
    REMAX = ONE / ABS( VR( II, KI ) )
    CALL DSCAL( N, REMAX, VR( 1, KI ), 1 )
  END IF
  *
ELSE
  *
  *      Complex right eigenvector.
  *
  *      Initial solve
  *      [ (T(KI-1,KI-1) T(KI-1,KI) ) - (WR + I* WI)]*X = 0.
  *      [ (T(KI,KI-1)   T(KI,KI)   )
  *
  IF( ABS( T( KI-1, KI ) ).GE.ABS( T( KI, KI-1 ) ) ) THEN
    WORK( KI-1+N ) = ONE
    WORK( KI+N2 ) = WI / T( KI-1, KI )
  ELSE
    WORK( KI-1+N ) = -WI / T( KI, KI-1 )
    WORK( KI+N2 ) = ONE
  END IF
  WORK( KI+N ) = ZERO
  WORK( KI-1+N2 ) = ZERO
  *
  *      Form right-hand side
  *
  DO 80 K = 1, KI - 2
    WORK( K+N ) = -WORK( KI-1+N )*T( K, KI-1 )
    WORK( K+N2 ) = -WORK( KI+N2 )*T( K, KI )
80  CONTINUE
  *
  *      Solve upper quasi-triangular system:
  *      (T(1:KI-2,1:KI-2) - (WR+i*WI))*X = SCALE*(WORK+i*WORK2)
  *
  JNXT = KI - 2
  DO 90 J = KI - 2, 1, -1
    IF( J.GT.JNXT )
      $      GO TO 90
    J1 = J
    J2 = J
    JNXT = J - 1
    IF( J.GT.1 ) THEN
      IF( T( J, J-1 ).NE.ZERO ) THEN
        J1 = J - 1
        JNXT = J - 2

```



```

      END IF
END IF

*
*
*      IF( J1.EQ.J2 ) THEN
*
*          1-by-1 diagonal block
*
*          CALL DLALN2( .FALSE., 1, 2, SMIN, ONE, T( J, J ),
$              LDT, ONE, ONE, WORK( J+N ), N, WR, WI,
$              X, 2, SCALE, XNORM, IERR )
*
*          Scale X(1,1) and X(1,2) to avoid overflow when
*          updating the right-hand side.
*
*          IF( XNORM.GT.ONE ) THEN
*              IF( WORK( J ).GT.BIGNUM / XNORM ) THEN
*                  X( 1, 1 ) = X( 1, 1 ) / XNORM
*                  X( 1, 2 ) = X( 1, 2 ) / XNORM
*                  SCALE = SCALE / XNORM
*              END IF
*          END IF
*
*          Scale if necessary
*
*          IF( SCALE.NE.ONE ) THEN
*              CALL DSCAL( KI, SCALE, WORK( 1+N ), 1 )
*              CALL DSCAL( KI, SCALE, WORK( 1+N2 ), 1 )
*          END IF
*          WORK( J+N ) = X( 1, 1 )
*          WORK( J+N2 ) = X( 1, 2 )
*
*          Update the right-hand side
*
*          CALL DAXPY( J-1, -X( 1, 1 ), T( 1, J ), 1,
$              WORK( 1+N ), 1 )
*          CALL DAXPY( J-1, -X( 1, 2 ), T( 1, J ), 1,
$              WORK( 1+N2 ), 1 )
*
*      ELSE
*
*          2-by-2 diagonal block
*
*          CALL DLALN2( .FALSE., 2, 2, SMIN, ONE,
$              T( J-1, J-1 ), LDT, ONE, ONE,
$              WORK( J-1+N ), N, WR, WI, X, 2, SCALE,
$              XNORM, IERR )
*
*          Scale X to avoid overflow when updating
*          the right-hand side.
*

```

```

IF( XNORM.GT.ONE ) THEN
  BETA = MAX( WORK( J-1 ), WORK( J ) )
  IF( BETA.GT.BIGNUM / XNORM ) THEN
    REC = ONE / XNORM
    X( 1, 1 ) = X( 1, 1 )*REC
    X( 1, 2 ) = X( 1, 2 )*REC
    X( 2, 1 ) = X( 2, 1 )*REC
    X( 2, 2 ) = X( 2, 2 )*REC
    SCALE = SCALE*REC
  END IF
END IF

*
*
*
Scale if necessary

IF( SCALE.NE.ONE ) THEN
  CALL DSCAL( KI, SCALE, WORK( 1+N ), 1 )
  CALL DSCAL( KI, SCALE, WORK( 1+N2 ), 1 )
END IF
WORK( J-1+N ) = X( 1, 1 )
WORK( J+N ) = X( 2, 1 )
WORK( J-1+N2 ) = X( 1, 2 )
WORK( J+N2 ) = X( 2, 2 )

*
*
*
Update the right-hand side

CALL DAXPY( J-2, -X( 1, 1 ), T( 1, J-1 ), 1,
$          WORK( 1+N ), 1 )
CALL DAXPY( J-2, -X( 2, 1 ), T( 1, J ), 1,
$          WORK( 1+N ), 1 )
CALL DAXPY( J-2, -X( 1, 2 ), T( 1, J-1 ), 1,
$          WORK( 1+N2 ), 1 )
CALL DAXPY( J-2, -X( 2, 2 ), T( 1, J ), 1,
$          WORK( 1+N2 ), 1 )

END IF
90 CONTINUE

*
*
*
Copy the vector x or Q*x to VR and normalize.

IF( .NOT.OVER ) THEN
  CALL DCOPY( KI, WORK( 1+N ), 1, VR( 1, IS-1 ), 1 )
  CALL DCOPY( KI, WORK( 1+N2 ), 1, VR( 1, IS ), 1 )

*
  EMAX = ZERO
  DO 100 K = 1, KI
    EMAX = MAX( EMAX, ABS( VR( K, IS-1 ) )+
$          ABS( VR( K, IS ) ) )
100 CONTINUE

*
REMAX = ONE / EMAX
CALL DSCAL( KI, REMAX, VR( 1, IS-1 ), 1 )

```

```

        CALL DSCAL( KI, REMAX, VR( 1, IS ), 1 )
*
        DO 110 K = KI + 1, N
            VR( K, IS-1 ) = ZERO
            VR( K, IS ) = ZERO
110      CONTINUE
*
        ELSE
*
            IF( KI.GT.2 ) THEN
                CALL DGEMV( 'N', N, KI-2, ONE, VR, LDVR,
$                   WORK( 1+N ), 1, WORK( KI-1+N ),
$                   VR( 1, KI-1 ), 1 )
                CALL DGEMV( 'N', N, KI-2, ONE, VR, LDVR,
$                   WORK( 1+N2 ), 1, WORK( KI+N2 ),
$                   VR( 1, KI ), 1 )
            ELSE
                CALL DSCAL( N, WORK( KI-1+N ), VR( 1, KI-1 ), 1 )
                CALL DSCAL( N, WORK( KI+N2 ), VR( 1, KI ), 1 )
            END IF
*
            EMAX = ZERO
            DO 120 K = 1, N
                EMAX = MAX( EMAX, ABS( VR( K, KI-1 ) )+
$                   ABS( VR( K, KI ) ) )
120      CONTINUE
            REMAX = ONE / EMAX
            CALL DSCAL( N, REMAX, VR( 1, KI-1 ), 1 )
            CALL DSCAL( N, REMAX, VR( 1, KI ), 1 )
        END IF
    END IF
*
    IS = IS - 1
    IF( IP.NE.0 )
$       IS = IS - 1
130    CONTINUE
        IF( IP.EQ.1 )
$           IP = 0
        IF( IP.EQ.-1 )
$           IP = 1
140    CONTINUE
    END IF
*
    IF( LEFTV ) THEN
*
        Compute left eigenvectors.
*
        IP = 0
        IS = 1
        DO 260 KI = 1, N

```

```

*
      IF( IP.EQ.-1 )
$         GO TO 250
      IF( KI.EQ.N )
$         GO TO 150
      IF( T( KI+1, KI ).EQ.ZERO )
$         GO TO 150
      IP = 1
*
150      CONTINUE
      IF( SOMEV ) THEN
          IF( .NOT.SELECT( KI ) )
$              GO TO 250
      END IF
*
*      Compute the KI-th eigenvalue (WR,WI).
*
      WR = T( KI, KI )
      WI = ZERO
      IF( IP.NE.0 )
$          WI = SQRT( ABS( T( KI, KI+1 ) ) ) *
$              SQRT( ABS( T( KI+1, KI ) ) )
      SMIN = MAX( ULP*( ABS( WR )+ABS( WI ) ), SMLNUM )
*
      IF( IP.EQ.0 ) THEN
*
*          Real left eigenvector.
*
          WORK( KI+N ) = ONE
*
*          Form right-hand side
*
          DO 160 K = KI + 1, N
              WORK( K+N ) = -T( KI, K )
160      CONTINUE
*
*          Solve the quasi-triangular system:
*          (T(KI+1:N,KI+1:N) - WR)*X = SCALE*WORK
*
          VMAX = ONE
          VCRIT = BIGNUM
*
          JNXT = KI + 1
          DO 170 J = KI + 1, N
              IF( J.LT.JNXT )
$                  GO TO 170
              J1 = J
              J2 = J
              JNXT = J + 1
              IF( J.LT.N ) THEN

```

```

      IF( T( J+1, J ).NE.ZERO ) THEN
        J2 = J + 1
        JNXT = J + 2
      END IF
    END IF

*
    IF( J1.EQ.J2 ) THEN
*
*      1-by-1 diagonal block
*
*      Scale if necessary to avoid overflow when forming
*      the right-hand side.
*
      IF( WORK( J ).GT.VCRIT ) THEN
        REC = ONE / VMAX
        CALL DSCAL( N-KI+1, REC, WORK( KI+N ), 1 )
        VMAX = ONE
        VCRIT = BIGNUM
      END IF

*
      WORK( J+N ) = WORK( J+N ) -
$          DDOT( J-KI-1, T( KI+1, J ), 1,
$          WORK( KI+1+N ), 1 )

*
*      Solve (T(J,J)-WR)'*X = WORK
*
      CALL DLALN2( .FALSE., 1, 1, SMIN, ONE, T( J, J ),
$          LDT, ONE, ONE, WORK( J+N ), N, WR,
$          ZERO, X, 2, SCALE, XNORM, IERR )

*
*      Scale if necessary
*
      IF( SCALE.NE.ONE )
$          CALL DSCAL( N-KI+1, SCALE, WORK( KI+N ), 1 )
      WORK( J+N ) = X( 1, 1 )
      VMAX = MAX( ABS( WORK( J+N ) ), VMAX )
      VCRIT = BIGNUM / VMAX

*
    ELSE
*
*      2-by-2 diagonal block
*
*      Scale if necessary to avoid overflow when forming
*      the right-hand side.
*
      BETA = MAX( WORK( J ), WORK( J+1 ) )
      IF( BETA.GT.VCRIT ) THEN
        REC = ONE / VMAX
        CALL DSCAL( N-KI+1, REC, WORK( KI+N ), 1 )
        VMAX = ONE

```

```

        VCRIT = BIGNUM
        END IF
*
        WORK( J+N ) = WORK( J+N ) -
$           DDOT( J-KI-1, T( KI+1, J ), 1,
$           WORK( KI+1+N ), 1 )
*
        WORK( J+1+N ) = WORK( J+1+N ) -
$           DDOT( J-KI-1, T( KI+1, J+1 ), 1,
$           WORK( KI+1+N ), 1 )
*
        Solve
*
*       [T(J,J)-WR   T(J,J+1)       ]'* X = SCALE*( WORK1 )
*       [T(J+1,J)   T(J+1,J+1)-WR]       ( WORK2 )
*
        CALL DLALN2( .TRUE., 2, 1, SMIN, ONE, T( J, J ),
$           LDT, ONE, ONE, WORK( J+N ), N, WR,
$           ZERO, X, 2, SCALE, XNORM, IERR )
*
        Scale if necessary
*
        IF( SCALE.NE.ONE )
$           CALL DSCAL( N-KI+1, SCALE, WORK( KI+N ), 1 )
        WORK( J+N ) = X( 1, 1 )
        WORK( J+1+N ) = X( 2, 1 )
*
        VMAX = MAX( ABS( WORK( J+N ) ),
$           ABS( WORK( J+1+N ) ), VMAX )
        VCRIT = BIGNUM / VMAX
*
        END IF
170      CONTINUE
*
        Copy the vector x or Q*x to VL and normalize.
*
        IF( .NOT.OVER ) THEN
            CALL DCOPY( N-KI+1, WORK( KI+N ), 1, VL( KI, IS ), 1 )
*
            II = IDAMAX( N-KI+1, VL( KI, IS ), 1 ) + KI - 1
            REMAX = ONE / ABS( VL( II, IS ) )
            CALL DSCAL( N-KI+1, REMAX, VL( KI, IS ), 1 )
*
            DO 180 K = 1, KI - 1
                VL( K, IS ) = ZERO
180          CONTINUE
*
        ELSE
*
            IF( KI.LT.N )
$           CALL DGEMV( 'N', N, N-KI, ONE, VL( 1, KI+1 ), LDVL,

```

```

$                                WORK( KI+1+N ), 1, WORK( KI+N ),
$                                VL( 1, KI ), 1 )
*
    II = IDAMAX( N, VL( 1, KI ), 1 )
    REMAX = ONE / ABS( VL( II, KI ) )
    CALL DSCAL( N, REMAX, VL( 1, KI ), 1 )
*
    END IF
*
ELSE
*
    Complex left eigenvector.
*
    Initial solve:
*      ((T(KI,KI)      T(KI,KI+1) )' - (WR - I* WI))*X = 0.
*      ((T(KI+1,KI) T(KI+1,KI+1))
*
    IF( ABS( T( KI, KI+1 ) ) .GE. ABS( T( KI+1, KI ) ) ) THEN
        WORK( KI+N ) = WI / T( KI, KI+1 )
        WORK( KI+1+N2 ) = ONE
    ELSE
        WORK( KI+N ) = ONE
        WORK( KI+1+N2 ) = -WI / T( KI+1, KI )
    END IF
    WORK( KI+1+N ) = ZERO
    WORK( KI+N2 ) = ZERO
*
*      Form right-hand side
*
    DO 190 K = KI + 2, N
        WORK( K+N ) = -WORK( KI+N ) * T( KI, K )
        WORK( K+N2 ) = -WORK( KI+1+N2 ) * T( KI+1, K )
190    CONTINUE
*
    Solve complex quasi-triangular system:
*      ( T(KI+2,N:KI+2,N) - (WR-i*WI) ) * X = WORK1+i*WORK2
*
    VMAX = ONE
    VCRIT = BIGNUM
*
    JNXT = KI + 2
    DO 200 J = KI + 2, N
        IF( J.LT.JNXT )
$            GO TO 200
        J1 = J
        J2 = J
        JNXT = J + 1
        IF( J.LT.N ) THEN
            IF( T( J+1, J ) .NE. ZERO ) THEN
                J2 = J + 1

```

```

        JNXT = J + 2
      END IF
    END IF

*
    IF( J1.EQ.J2 ) THEN
*
*       1-by-1 diagonal block
*
*       Scale if necessary to avoid overflow when
*       forming the right-hand side elements.
*
      IF( WORK( J ).GT.VCRIT ) THEN
        REC = ONE / VMAX
        CALL DSCAL( N-KI+1, REC, WORK( KI+N ), 1 )
        CALL DSCAL( N-KI+1, REC, WORK( KI+N2 ), 1 )
        VMAX = ONE
        VCRIT = BIGNUM
      END IF

*
      WORK( J+N ) = WORK( J+N ) -
$          DDOT( J-KI-2, T( KI+2, J ), 1,
$          WORK( KI+2+N ), 1 )
      WORK( J+N2 ) = WORK( J+N2 ) -
$          DDOT( J-KI-2, T( KI+2, J ), 1,
$          WORK( KI+2+N2 ), 1 )

*
*       Solve (T(J,J)-(WR-i*WI))*(X11+i*X12)= WK+I*WK2
*
      CALL DLALN2( .FALSE., 1, 2, SMIN, ONE, T( J, J ),
$          LDT, ONE, ONE, WORK( J+N ), N, WR,
$          -WI, X, 2, SCALE, XNORM, IERR )

*
*       Scale if necessary
*
      IF( SCALE.NE.ONE ) THEN
        CALL DSCAL( N-KI+1, SCALE, WORK( KI+N ), 1 )
        CALL DSCAL( N-KI+1, SCALE, WORK( KI+N2 ), 1 )
      END IF
      WORK( J+N ) = X( 1, 1 )
      WORK( J+N2 ) = X( 1, 2 )
      VMAX = MAX( ABS( WORK( J+N ) ),
$          ABS( WORK( J+N2 ) ), VMAX )
      VCRIT = BIGNUM / VMAX

*
    ELSE

*
*       2-by-2 diagonal block
*
*       Scale if necessary to avoid overflow when forming
*       the right-hand side elements.

```



```

*
      BETA = MAX( WORK( J ), WORK( J+1 ) )
      IF( BETA.GT.VCRIT ) THEN
        REC = ONE / VMAX
        CALL DSCAL( N-KI+1, REC, WORK( KI+N ), 1 )
        CALL DSCAL( N-KI+1, REC, WORK( KI+N2 ), 1 )
        VMAX = ONE
        VCRIT = BIGNUM
      END IF

*
      WORK( J+N ) = WORK( J+N ) -
$          DDOT( J-KI-2, T( KI+2, J ), 1,
$          WORK( KI+2+N ), 1 )

*
      WORK( J+N2 ) = WORK( J+N2 ) -
$          DDOT( J-KI-2, T( KI+2, J ), 1,
$          WORK( KI+2+N2 ), 1 )

*
      WORK( J+1+N ) = WORK( J+1+N ) -
$          DDOT( J-KI-2, T( KI+2, J+1 ), 1,
$          WORK( KI+2+N ), 1 )

*
      WORK( J+1+N2 ) = WORK( J+1+N2 ) -
$          DDOT( J-KI-2, T( KI+2, J+1 ), 1,
$          WORK( KI+2+N2 ), 1 )

*
*      Solve 2-by-2 complex linear equation
*      ([T(j,j)  T(j,j+1) ]'-(wr-i*wi)*I)*X = SCALE*B
*      ([T(j+1,j) T(j+1,j+1)]          )
*
      CALL DLALN2( .TRUE., 2, 2, SMIN, ONE, T( J, J ),
$          LDT, ONE, ONE, WORK( J+N ), N, WR,
$          -WI, X, 2, SCALE, XNORM, IERR )

*
*      Scale if necessary
*
      IF( SCALE.NE.ONE ) THEN
        CALL DSCAL( N-KI+1, SCALE, WORK( KI+N ), 1 )
        CALL DSCAL( N-KI+1, SCALE, WORK( KI+N2 ), 1 )
      END IF
      WORK( J+N ) = X( 1, 1 )
      WORK( J+N2 ) = X( 1, 2 )
      WORK( J+1+N ) = X( 2, 1 )
      WORK( J+1+N2 ) = X( 2, 2 )
      VMAX = MAX( ABS( X( 1, 1 ) ), ABS( X( 1, 2 ) ),
$          ABS( X( 2, 1 ) ), ABS( X( 2, 2 ) ), VMAX )
      VCRIT = BIGNUM / VMAX

*
      END IF
200      CONTINUE

```

```

*
*      Copy the vector x or Q*x to VL and normalize.
*
210      CONTINUE
      IF( .NOT.OVER ) THEN
          CALL DCOPY( N-KI+1, WORK( KI+N ), 1, VL( KI, IS ), 1 )
          CALL DCOPY( N-KI+1, WORK( KI+N2 ), 1, VL( KI, IS+1 ),
$              1 )
*
          EMAX = ZERO
          DO 220 K = KI, N
              EMAX = MAX( EMAX, ABS( VL( K, IS ) )+
$                  ABS( VL( K, IS+1 ) ) )
220      CONTINUE
          REMAX = ONE / EMAX
          CALL DSCAL( N-KI+1, REMAX, VL( KI, IS ), 1 )
          CALL DSCAL( N-KI+1, REMAX, VL( KI, IS+1 ), 1 )
*
          DO 230 K = 1, KI - 1
              VL( K, IS ) = ZERO
              VL( K, IS+1 ) = ZERO
230      CONTINUE
          ELSE
              IF( KI.LT.N-1 ) THEN
                  CALL DGEMV( 'N', N, N-KI-1, ONE, VL( 1, KI+2 ),
$                      LDVL, WORK( KI+2+N ), 1, WORK( KI+N ),
$                      VL( 1, KI ), 1 )
                  CALL DGEMV( 'N', N, N-KI-1, ONE, VL( 1, KI+2 ),
$                      LDVL, WORK( KI+2+N2 ), 1,
$                      WORK( KI+1+N2 ), VL( 1, KI+1 ), 1 )
              ELSE
                  CALL DSCAL( N, WORK( KI+N ), VL( 1, KI ), 1 )
                  CALL DSCAL( N, WORK( KI+1+N2 ), VL( 1, KI+1 ), 1 )
              END IF
*
          EMAX = ZERO
          DO 240 K = 1, N
              EMAX = MAX( EMAX, ABS( VL( K, KI ) )+
$                  ABS( VL( K, KI+1 ) ) )
240      CONTINUE
          REMAX = ONE / EMAX
          CALL DSCAL( N, REMAX, VL( 1, KI ), 1 )
          CALL DSCAL( N, REMAX, VL( 1, KI+1 ), 1 )
*
      END IF
*
      END IF
*
      IS = IS + 1
      IF( IP.NE.0 )

```

```

$      IS = IS + 1
250    CONTINUE
      IF( IP.EQ.-1 )
$      IP = 0
      IF( IP.EQ.1 )
$      IP = -1
*
260    CONTINUE
*
      END IF
*
      RETURN
*
*      End of DTREVC
*
      END

```

— LAPACK dtrevc —

```

(let* ((zero 0.0) (one 1.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one))
  (defun dtrevc (side howmny select n t$ ldt vl ldvl vr ldvr mm m work info)
    (declare (type (simple-array double-float (*)) work vr vl t$)
              (type fixnum info m mm ldvr ldvl ldt n)
              (type (simple-array (member t nil) (*)) select)
              (type character howmny side))
    (f2cl-lib:with-multi-array-data
      ((side character side-%data% side-%offset%)
       (howmny character howmny-%data% howmny-%offset%)
       (select (member t nil) select-%data% select-%offset%)
       (t$ double-float t$-%data% t$-%offset%)
       (vl double-float vl-%data% vl-%offset%)
       (vr double-float vr-%data% vr-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((x (make-array 4 :element-type 'double-float)) (beta 0.0)
              (bignum 0.0) (emax 0.0) (ovfl 0.0) (rec 0.0) (remax 0.0)
              (scale 0.0) (smin 0.0) (smlnum 0.0) (ulp 0.0) (unfl 0.0)
              (vcrit 0.0) (vmax 0.0) (wi 0.0) (wr 0.0) (xnorm 0.0) (i 0)
              (ierr 0) (ii 0) (ip 0) (is 0) (j 0) (j1 0) (j2 0) (jnxt 0) (k 0)
              (ki 0) (n2 0) (allv nil) (bothv nil) (leftv nil) (over nil)
              (pair nil) (rightv nil) (somev nil) (sqrt$ 0.0f0))
              (declare (type (single-float) sqrt$)
                        (type (simple-array double-float (4)) x)
                        (type (double-float) beta bignum emax ovfl rec remax scale
                               smin smlnum ulp unfl vcrit vmax wi wr

```

```

                                xnorm)
      (type fixnum i ierr ii ip is j j1 j2 jnxt k ki
                                n2)
      (type (member t nil) allv bothv leftv over pair rightv
                                somev))
      (setf bothv (char-equal side #\B))
      (setf rightv (or (char-equal side #\R) bothv))
      (setf leftv (or (char-equal side #\L) bothv))
      (setf allv (char-equal howmny #\A))
      (setf over (char-equal howmny #\B))
      (setf somev (char-equal howmny #\S))
      (setf info 0)
      (cond
        ((and (not rightv) (not leftv))
         (setf info -1))
        ((and (not allv) (not over) (not somev))
         (setf info -2))
        ((< n 0)
         (setf info -4))
        ((< ldt (max (the fixnum 1) (the fixnum n)))
         (setf info -6))
        ((or (< ldvl 1) (and leftv (< ldvl n)))
         (setf info -8))
        ((or (< ldvr 1) (and rightv (< ldvr n)))
         (setf info -10))
        (t
         (cond
           (somev
            (setf m 0)
            (setf pair nil)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          (> j n) nil)
            (tagbody
             (cond
              (pair
               (setf pair nil)
               (setf (f2cl-lib:fref select-%data%
                                   (j)
                                   ((1 *))
                                   select-%offset%)
                     nil)))
             (t
              (cond
                ((< j n)
                 (cond
                  ((=
                    (f2cl-lib:fref t$
                                   ((f2cl-lib:int-add j 1) j)
                                   ((1 ldt) (1 *)))
                     zero)

```

```

      (if
        (f2cl-lib:fref select-%data%
                      (j)
                      ((1 *))
                      select-%offset%)
        (setf m (f2cl-lib:int-add m 1))))
    (t
     (setf pair t)
     (cond
      ((or (f2cl-lib:fref select (j) ((1 *))
                                (f2cl-lib:fref select
                                              ((f2cl-lib:int-add j 1))
                                              ((1 *))))
          (setf (f2cl-lib:fref select-%data%
                                (j)
                                ((1 *))
                                select-%offset%)
                 t)
          (setf m (f2cl-lib:int-add m 2))))))
    (t
     (if
      (f2cl-lib:fref select-%data%
                    (n)
                    ((1 *))
                    select-%offset%)
      (setf m (f2cl-lib:int-add m 1))))))
  (t
   (setf m n))
  (cond
   ((< mm m)
    (setf info -11))))
(cond
 ((/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "DTREVC" (f2cl-lib:int-sub info))
  (go end_label)))
(if (= n 0) (go end_label))
(setf unfl (dlamch "Safe minimum"))
(setf ovfl (/ one unfl))
(multiple-value-bind (var-0 var-1)
  (dlabad unfl ovfl))
(declare (ignore))
(setf unfl var-0)
(setf ovfl var-1)
(setf ulp (dlamch "Precision"))
(setf smlnum (* unfl (/ n ulp)))
(setf bignum (/ (- one ulp) smlnum))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) zero)
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))

```

```

        (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%) zero)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i (f2cl-lib:int-add j (f2cl-lib:int-sub 1))) nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
      (+
        (f2cl-lib:fref work-%data% (j) ((1 *)) work-%offset%)
        (abs
          (f2cl-lib:fref t$-%data%
            (i j)
            ((1 ldt) (1 *))
            t$-%offset%))))))
(setf n2 (f2cl-lib:int-mul 2 n))
(cond
  (rightv
    (setf ip 0)
    (setf is m)
    (f2cl-lib:fdo (ki n (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
      (> ki 1) nil)
    (tagbody
      (if (= ip 1) (go label130))
      (if (= ki 1) (go label40))
      (if
        (=
          (f2cl-lib:fref t$-%data%
            (ki (f2cl-lib:int-sub ki 1))
            ((1 ldt) (1 *))
            t$-%offset%)
          zero)
        (go label40))
      (setf ip -1)
label40
      (cond
        (somev
          (cond
            ((= ip 0)
              (if
                (not
                  (f2cl-lib:fref select-%data%
                    (ki)
                    ((1 *))
                    select-%offset%))
                (go label130))))
            (t
              (if
                (not
                  (f2cl-lib:fref select-%data%
                    ((f2cl-lib:int-sub ki 1))

```

```

                                ((1 *))
                                select-%offset%)
      (go label130))))))
(setf wr
  (f2cl-lib:fref t$-%data%
    (ki ki)
    ((1 ldt) (1 *))
    t$-%offset%))
(setf wi zero)
(if (/= ip 0)
  (setf wi
    (*
      (f2cl-lib:fsqrt
        (abs
          (f2cl-lib:fref t$-%data%
            (ki (f2cl-lib:int-sub ki 1))
            ((1 ldt) (1 *))
            t$-%offset%)))
      (f2cl-lib:fsqrt
        (abs
          (f2cl-lib:fref t$-%data%
            ((f2cl-lib:int-sub ki 1) ki)
            ((1 ldt) (1 *))
            t$-%offset%))))))
(setf smin (max (* ulp (+ (abs wr) (abs wi))) smlnum))
(cond
  ((= ip 0)
    (setf (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-add ki n))
      ((1 *))
      work-%offset%)
      one)
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
      (> k
        (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add k n))
        ((1 *))
        work-%offset%)
        (-
          (f2cl-lib:fref t$-%data%
            (k ki)
            ((1 ldt) (1 *))
            t$-%offset%))))))
  (setf jnxt (f2cl-lib:int-sub ki 1))
  (f2cl-lib:fdo (j (f2cl-lib:int-add ki (f2cl-lib:int-sub 1))
    (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
    (> j 1) nil)

```

```

(tagbody
  (if (> j jnxt) (go label60))
  (setf j1 j)
  (setf j2 j)
  (setf jnxt (f2cl-lib:int-sub j 1))
  (cond
    ((> j 1)
     (cond
       ((/=
         (f2cl-lib:fref t$
                       (j
                        (f2cl-lib:int-add j
                                           (f2cl-lib:int-sub
                                            1)))
                        ((1 ldt) (1 *)))
          zero)
        (setf j1 (f2cl-lib:int-sub j 1))
        (setf jnxt (f2cl-lib:int-sub j 2))))))
    ((= j1 j2)
     (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10 var-11 var-12 var-13 var-14
        var-15 var-16 var-17)
       (dlaln2 nil 1 1 smin one
        (f2cl-lib:array-slice t$
                              double-float
                              (j j)
                              ((1 ldt) (1 *)))
          ldt one one
          (f2cl-lib:array-slice work
                                double-float
                                ((+ j n))
                                ((1 *)))
          n wr zero x 2 scale xnorm ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                       var-6 var-7 var-8 var-9 var-10
                       var-11 var-12 var-13 var-14))
      (setf scale var-15)
      (setf xnorm var-16)
      (setf ierr var-17))
     (cond
       ((> xnorm one)
        (cond
          ((> (f2cl-lib:fref work (j) ((1 *)))
              (f2cl-lib:f2cl/ bignum xnorm))
           (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
                  (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
                     xnorm))
           (setf scale (/ scale xnorm))))))

```



```

(if (/= scale one)
  (dscal ki scale
    (f2cl-lib:array-slice work
      double-float
      ((+ 1 n))
      ((1 *)))
    1))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 j)
    ((1 ldt) (1 *)))
  1
  (f2cl-lib:array-slice work
    double-float
    ((+ 1 n))
    ((1 *)))
  1))
(t
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12 var-13 var-14
     var-15 var-16 var-17)
    (dlaln2 nil 2 1 smin one
      (f2cl-lib:array-slice t$
        double-float
        ((+ j (f2cl-lib:int-sub 1))
         (f2cl-lib:int-sub j 1))
        ((1 ldt) (1 *)))
      ldt one one
      (f2cl-lib:array-slice work
        double-float
        ((+ j
          (f2cl-lib:int-sub 1)
          n))
        ((1 *)))
      n wr zero x 2 scale xnorm ierr)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
      var-6 var-7 var-8 var-9 var-10
      var-11 var-12 var-13 var-14))
    (setf scale var-15)
    (setf xnorm var-16)
    (setf ierr var-17))
  (cond

```

```

(> xnorm one)
(setf beta
  (max
    (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-sub j 1))
      ((1 *))
      work-%offset%)
    (f2cl-lib:fref work-%data%
      (j)
      ((1 *))
      work-%offset%)))
(cond
  (> beta (f2cl-lib:f2cl/ bignum xnorm))
  (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
    (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
        xnorm))
  (setf (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
    (/ (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
        xnorm))
  (setf scale (/ scale xnorm))))
(if (/= scale one)
  (dscal ki scale
    (f2cl-lib:array-slice work
      double-float
      ((+ 1 n))
      ((1 *)))
    1))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub j 1)
    n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n)
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 2)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 (f2cl-lib:int-sub j 1))
    ((1 ldt) (1 *)))
  1
  (f2cl-lib:array-slice work
    double-float
    ((+ 1 n))
    ((1 *)))

```

```

1)
(daxpy (f2cl-lib:int-sub j 2)
(- (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(f2cl-lib:array-slice t$
double-float
(1 j)
((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
double-float
((+ 1 n))
((1 *)))

1)))
label60))
(cond
((not over)
(dcopy ki
(f2cl-lib:array-slice work
double-float
((+ 1 n))
((1 *)))

1
(f2cl-lib:array-slice vr
double-float
(1 is)
((1 ldvr) (1 *)))

1)
(setf ii
(idamax ki
(f2cl-lib:array-slice vr
double-float
(1 is)
((1 ldvr) (1 *)))

1))
(setf remax
(/ one
(abs
(f2cl-lib:fref vr-%data%
(ii is)
((1 ldvr) (1 *))
vr-%offset%))))))
(dscal ki remax
(f2cl-lib:array-slice vr
double-float
(1 is)
((1 ldvr) (1 *)))

1)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
(f2cl-lib:int-add k 1))
(> k n) nil)

```

```

(tagbody
  (setf (f2cl-lib:fref vr-%data%
                      (k is)
                      ((1 ldvr) (1 *))
                      vr-%offset%)
        zero))))
(t
  (if (> ki 1)
    (dgemv "N" n (f2cl-lib:int-sub ki 1) one vr ldvr
      (f2cl-lib:array-slice work
                            double-float
                            ((+ 1 n))
                            ((1 *)))
      1
      (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add ki n))
                      ((1 *))
                      work-%offset%)
      (f2cl-lib:array-slice vr
                            double-float
                            (1 ki)
                            ((1 ldvr) (1 *)))
      1))
  (setf ii
    (idamax n
      (f2cl-lib:array-slice vr
                            double-float
                            (1 ki)
                            ((1 ldvr) (1 *)))
      1))
  (setf remax
    (/ one
      (abs
        (f2cl-lib:fref vr-%data%
                        (ii ki)
                        ((1 ldvr) (1 *))
                        vr-%offset%))))
  (dscal n remax
    (f2cl-lib:array-slice vr
                          double-float
                          (1 ki)
                          ((1 ldvr) (1 *)))
    1))))
(t
  (cond
    ((>=
      (abs
        (f2cl-lib:fref t$
                        ((f2cl-lib:int-add ki
                          (f2cl-lib:int-sub 1))

```

```

                                ki)
                                ((1 ldt) (1 *))))
(abs
  (f2cl-lib:fref t$
    (ki
      (f2cl-lib:int-add ki
        (f2cl-lib:int-sub 1)))
    ((1 ldt) (1 *))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub ki 1)
    n))
  ((1 *))
  work-%offset%)
  one)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add ki n2))
  ((1 *))
  work-%offset%)
  (/ wi
    (f2cl-lib:fref t$-%data%
      ((f2cl-lib:int-sub ki 1) ki)
      ((1 ldt) (1 *))
      t$-%offset%))))
(t
  (setf (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add
      (f2cl-lib:int-sub ki 1)
      n))
    ((1 *))
    work-%offset%)
    (/ (- wi)
      (f2cl-lib:fref t$-%data%
        (ki (f2cl-lib:int-sub ki 1))
        ((1 ldt) (1 *))
        t$-%offset%))))
  (setf (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add ki n2))
    ((1 *))
    work-%offset%)
    one)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add ki n))
  ((1 *))
  work-%offset%)
  zero)
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub ki 1)
    n2))

```

```

                                ((1 *))
                                work-%offset%)
                                zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k
                (f2cl-lib:int-add ki (f2cl-lib:int-sub 2)))
              nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add k n))
                      ((1 *))
                      work-%offset%)
        (*
          (-
            (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add
                           (f2cl-lib:int-sub ki 1)
                           n))
                          ((1 *))
                          work-%offset%))
            (f2cl-lib:fref t$-%data%
                          (k (f2cl-lib:int-sub ki 1))
                          ((1 ldt) (1 *))
                          t$-%offset%)))
        (setf (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add k n2))
                          ((1 *))
                          work-%offset%)
              (*
                (-
                  (f2cl-lib:fref work-%data%
                                ((f2cl-lib:int-add ki n2))
                                ((1 *))
                                work-%offset%))
                  (f2cl-lib:fref t$-%data%
                                (k ki)
                                ((1 ldt) (1 *))
                                t$-%offset%))))))
(setf jnxt (f2cl-lib:int-sub ki 2))
(f2cl-lib:fdo (j (f2cl-lib:int-add ki (f2cl-lib:int-sub 2))
              (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
              ((> j 1) nil)
(tagbody
  (if (> j jnxt) (go label90))
  (setf j1 j)
  (setf j2 j)
  (setf jnxt (f2cl-lib:int-sub j 1))
  (cond
    ((> j 1)
     (cond

```

```

( (/ =
  (f2cl-lib:fref t$
    (j
      (f2cl-lib:int-add j
        (f2cl-lib:int-sub
          1)))
      ((1 ldt) (1 *)))
  zero)
(setf j1 (f2cl-lib:int-sub j 1))
(setf jnxt (f2cl-lib:int-sub j 2))))))
(cond
  ((= j1 j2)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10 var-11 var-12 var-13 var-14
       var-15 var-16 var-17)
      (dlaln2 nil 1 2 smin one
        (f2cl-lib:array-slice t$
          double-float
          (j j)
          ((1 ldt) (1 *)))
        ldt one one
        (f2cl-lib:array-slice work
          double-float
          ((+ j n))
          ((1 *)))
        n wr wi x 2 scale xnorm ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
        var-6 var-7 var-8 var-9 var-10
        var-11 var-12 var-13 var-14))
      (setf scale var-15)
      (setf xnorm var-16)
      (setf ierr var-17))
    (cond
      ((> xnorm one)
        (cond
          ((> (f2cl-lib:fref work (j) ((1 *)))
            (f2cl-lib:f2cl/ bignum xnorm))
            (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
              (/ (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
                xnorm))
            (setf (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
              (/ (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
                xnorm))
            (setf scale (/ scale xnorm))))))
      (cond
        ((/= scale one)
          (dscal ki scale
            (f2cl-lib:array-slice work
              double-float

```

```

                                ((+ 1 n))
                                ((1 *)))

1)
(dscal ki scale
  (f2cl-lib:array-slice work
    double-float
    ((+ 1 n2))
    ((1 *)))

1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n))
  ((1 *))
  work-%offset%))
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n2))
  ((1 *))
  work-%offset%))
  (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 j)
    ((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n))
  ((1 *)))

1)
(daxpy (f2cl-lib:int-sub j 1)
  (- (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 j)
    ((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n2))
  ((1 *)))

1))
(t
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11 var-12 var-13 var-14
     var-15 var-16 var-17)
    (dlaln2 nil 2 2 smin one
      (f2cl-lib:array-slice t$

```



```

double-float
((+ j (f2cl-lib:int-sub 1))
 (f2cl-lib:int-sub j 1))
((1 ldt) (1 *)))

ldt one one
(f2cl-lib:array-slice work
 double-float
 ((+ j
  (f2cl-lib:int-sub 1)
  n))
 ((1 *)))
n wr wi x 2 scale xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
 var-6 var-7 var-8 var-9 var-10
 var-11 var-12 var-13 var-14))
(setf scale var-15)
(setf xnorm var-16)
(setf ierr var-17))
(cond
 (> xnorm one)
 (setf beta
  (max
   (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-sub j 1))
    ((1 *))
    work-%offset%)
   (f2cl-lib:fref work-%data%
    (j)
    ((1 *))
    work-%offset%)))
 (cond
  (> beta (f2cl-lib:f2cl/ bignum xnorm))
  (setf rec (/ one xnorm))
  (setf (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
   (* (f2cl-lib:fref x (1 1) ((1 2) (1 2)))
      rec))
  (setf (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
   (* (f2cl-lib:fref x (1 2) ((1 2) (1 2)))
      rec))
  (setf (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
   (* (f2cl-lib:fref x (2 1) ((1 2) (1 2)))
      rec))
  (setf (f2cl-lib:fref x (2 2) ((1 2) (1 2)))
   (* (f2cl-lib:fref x (2 2) ((1 2) (1 2)))
      rec))
  (setf scale (* scale rec))))))
(cond
 (/= scale one)
 (dscal ki scale
  (f2cl-lib:array-slice work

```

```

double-float
((+ 1 n))
((1 *)))

1)
(dscal ki scale
  (f2cl-lib:array-slice work
    double-float
    ((+ 1 n2))
    ((1 *)))

1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub j 1)
    n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n)
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add
    (f2cl-lib:int-sub j 1)
    n2))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n2)
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
(daxpy (f2cl-lib:int-sub j 2)
  (- (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 (f2cl-lib:int-sub j 1))
    ((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n))
  ((1 *)))

1)
(daxpy (f2cl-lib:int-sub j 2)
  (- (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float

```

```

                                (1 j)
                                ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n))
  ((1 *)))

1)
(daxpy (f2cl-lib:int-sub j 2)
  (- (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 (f2cl-lib:int-sub j 1))
    ((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n2))
  ((1 *)))

1)
(daxpy (f2cl-lib:int-sub j 2)
  (- (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
  (f2cl-lib:array-slice t$
    double-float
    (1 j)
    ((1 ldt) (1 *)))

1
(f2cl-lib:array-slice work
  double-float
  ((+ 1 n2))
  ((1 *)))

1)))
label90))

(cond
  ((not over)
   (dcopy ki
    (f2cl-lib:array-slice work
      double-float
      ((+ 1 n))
      ((1 *)))

    1
    (f2cl-lib:array-slice vr
      double-float
      (1 (f2cl-lib:int-sub is 1))
      ((1 ldvr) (1 *)))

    1)
   (dcopy ki
    (f2cl-lib:array-slice work
      double-float
      ((+ 1 n2))

```

```

((1 *)))
1
(f2cl-lib:array-slice vr
                      double-float
                      (1 is)
                      ((1 ldvr) (1 *)))
1)
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              (> k ki) nil)
(tagbody
  (setf emax
        (max emax
              (+
                (abs
                  (f2cl-lib:fref vr-%data%
                                (k
                                  (f2cl-lib:int-sub is
                                    1))
                                ((1 ldvr) (1 *))
                                vr-%offset%))
                (abs
                  (f2cl-lib:fref vr-%data%
                                (k is)
                                ((1 ldvr) (1 *))
                                vr-%offset%))))))
(setf remax (/ one emax))
(dscal ki remax
  (f2cl-lib:array-slice vr
                        double-float
                        (1 (f2cl-lib:int-sub is 1))
                        ((1 ldvr) (1 *)))
  1)
(dscal ki remax
  (f2cl-lib:array-slice vr
                        double-float
                        (1 is)
                        ((1 ldvr) (1 *)))
  1)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                (f2cl-lib:int-add k 1))
              (> k n) nil)
(tagbody
  (setf (f2cl-lib:fref vr-%data%
                      (k (f2cl-lib:int-sub is 1))
                      ((1 ldvr) (1 *))
                      vr-%offset%)
        zero)
  (setf (f2cl-lib:fref vr-%data%
                      (k is)

```

```

                                ((1 ldvr) (1 *))
                                vr-%offset%)
                                zero))))
(t
(cond
  (> ki 2)
    (dgemv "N" n (f2cl-lib:int-sub ki 2) one vr ldvr
      (f2cl-lib:array-slice work
        double-float
        ((+ 1 n))
        ((1 *)))
      1
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add
          (f2cl-lib:int-sub ki 1)
          n))
        ((1 *))
        work-%offset%)
      (f2cl-lib:array-slice vr
        double-float
        (1 (f2cl-lib:int-sub ki 1))
        ((1 ldvr) (1 *)))
      1)
    (dgemv "N" n (f2cl-lib:int-sub ki 2) one vr ldvr
      (f2cl-lib:array-slice work
        double-float
        ((+ 1 n2))
        ((1 *)))
      1
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add ki n2))
        ((1 *))
        work-%offset%)
      (f2cl-lib:array-slice vr
        double-float
        (1 ki)
        ((1 ldvr) (1 *)))
      1))
(t
(dscal n
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add
      (f2cl-lib:int-sub ki 1)
      n))
    ((1 *))
    work-%offset%)
  (f2cl-lib:array-slice vr
    double-float
    (1 (f2cl-lib:int-sub ki 1))
    ((1 ldvr) (1 *)))
  1))

```

```

1)
(dscal n
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add ki n2))
    ((1 *))
    work-%offset%)
  (f2cl-lib:array-slice vr
    double-float
    (1 ki)
    ((1 ldvr) (1 *)))
1)))
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k n) nil)
(tagbody
  (setf emax
    (max emax
      (+
        (abs
          (f2cl-lib:fref vr-%data%
            (k
              (f2cl-lib:int-sub ki
                1))
            ((1 ldvr) (1 *))
            vr-%offset%))
        (abs
          (f2cl-lib:fref vr-%data%
            (k ki)
            ((1 ldvr) (1 *))
            vr-%offset%))))))
  (setf remax (/ one emax))
  (dscal n remax
    (f2cl-lib:array-slice vr
      double-float
      (1 (f2cl-lib:int-sub ki 1))
      ((1 ldvr) (1 *)))
    1)
  (dscal n remax
    (f2cl-lib:array-slice vr
      double-float
      (1 ki)
      ((1 ldvr) (1 *)))
    1))))
(setf is (f2cl-lib:int-sub is 1))
(if (/= ip 0) (setf is (f2cl-lib:int-sub is 1)))
label130
(if (= ip 1) (setf ip 0))
(if (= ip -1) (setf ip 1))))
(cond
  (leftv

```

```

(setf ip 0)
(setf is 1)
(f2cl-lib:fdo (ki 1 (f2cl-lib:int-add ki 1))
              (> ki n) nil)
(tagbody
 (if (= ip -1) (go label250))
 (if (= ki n) (go label150))
 (if
  (=
   (f2cl-lib:fref t$-%data%
                   ((f2cl-lib:int-add ki 1) ki)
                   ((1 ldt) (1 *)))
   t$-%offset%)
  zero)
 (go label150))
label150 (setf ip 1)
(cond
 (somev
  (if
   (not
    (f2cl-lib:fref select-%data% (ki) ((1 *)) select-%offset%))
    (go label250))))
 (setf wr
  (f2cl-lib:fref t$-%data%
                  (ki ki)
                  ((1 ldt) (1 *)))
  t$-%offset%))
(setf wi zero)
(if (/= ip 0)
  (setf wi
  (*
   (f2cl-lib:fsqrt
    (abs
     (f2cl-lib:fref t$-%data%
                     (ki (f2cl-lib:int-add ki 1))
                     ((1 ldt) (1 *)))
     t$-%offset%)))
   (f2cl-lib:fsqrt
    (abs
     (f2cl-lib:fref t$-%data%
                     ((f2cl-lib:int-add ki 1) ki)
                     ((1 ldt) (1 *)))
     t$-%offset%))))))
(setf smin (max (* ulp (+ (abs wr) (abs wi))) smlnum))
(cond
 ((= ip 0)
  (setf (f2cl-lib:fref work-%data%
                       ((f2cl-lib:int-add ki n)
                        (1 *)))
  )

```

```

                                work-%offset%)
                                one)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                (f2cl-lib:int-add k 1))
              ((> k n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add k n))
                      ((1 *))
                      work-%offset%))
        (-
          (f2cl-lib:fref t$-%data%
                        (ki k)
                        ((1 ldt) (1 *))
                        t$-%offset%))))))
(setf vmax one)
(setf vcrit bignum)
(setf jnxt (f2cl-lib:int-add ki 1))
(f2cl-lib:fdo (j (f2cl-lib:int-add ki 1)
                (f2cl-lib:int-add j 1))
              ((> j n) nil)
(tagbody
  (if (< j jnxt) (go label170))
  (setf j1 j)
  (setf j2 j)
  (setf jnxt (f2cl-lib:int-add j 1))
  (cond
    ((< j n)
     (cond
      ((/=
        (f2cl-lib:fref t$
                      ((f2cl-lib:int-add j 1) j)
                      ((1 ldt) (1 *)))
         zero)
       (setf j2 (f2cl-lib:int-add j 1))
       (setf jnxt (f2cl-lib:int-add j 2))))))
    (cond
      ((= j1 j2)
       (cond
        ((> (f2cl-lib:fref work (j) ((1 *))) vcrit)
         (setf rec (/ one vmax))
         (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
                  rec
                  (f2cl-lib:array-slice work
                                         double-float
                                         ((+ ki n))
                                         ((1 *)))
                  1)
         (setf vmax one)
         (setf vcrit bignum))))

```



```

(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j n))
                    ((1 *))
                    work-%offset%))
(-
  (f2cl-lib:fref work-%data%
                ((f2cl-lib:int-add j n))
                ((1 *))
                work-%offset%))
  (ddot (f2cl-lib:int-sub j ki 1)
        (f2cl-lib:array-slice t$
                              double-float
                              ((+ ki 1) j)
                              ((1 ldt) (1 *)))
        1
        (f2cl-lib:array-slice work
                              double-float
                              ((+ ki 1 n))
                              ((1 *)))
        1)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11 var-12 var-13 var-14
   var-15 var-16 var-17)
  (dlaln2 nil 1 1 smin one
          (f2cl-lib:array-slice t$
                                double-float
                                (j j)
                                ((1 ldt) (1 *)))
          ldt one one
          (f2cl-lib:array-slice work
                                double-float
                                ((+ j n))
                                ((1 *)))
          n wr zero x 2 scale xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                  var-6 var-7 var-8 var-9 var-10
                  var-11 var-12 var-13 var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(if (/= scale one)
  (dscal
   (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
   scale
   (f2cl-lib:array-slice work
                         double-float
                         ((+ ki n))
                         ((1 *)))
   1))
1))

```

```

(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j n))
                    ((1 *))
                    work-%offset%))
(f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf vmax
  (max
    (abs
      (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j n))
                    ((1 *))
                    work-%offset%))
    vmax))
(setf vcrit (/ bignum vmax))
(t
  (setf beta
    (max
      (f2cl-lib:fref work-%data%
                    (j)
                    ((1 *))
                    work-%offset%))
      (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j 1))
                    ((1 *))
                    work-%offset%)))
    (cond
      ((> beta vcrit)
        (setf rec (/ one vmax))
        (dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
          rec
          (f2cl-lib:array-slice work
                                double-float
                                ((+ ki n))
                                ((1 *)))
          1)
        (setf vmax one)
        (setf vcrit bignum)))
    (setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j n))
                    ((1 *))
                    work-%offset%))
      (-
        (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add j n))
                    ((1 *))
                    work-%offset%))
        (ddot (f2cl-lib:int-sub j ki 1)
          (f2cl-lib:array-slice t$
                                double-float
                                ((+ ki 1) j)

```

```

                                ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice work
                        double-float
                        ((+ ki 1 n))
                        ((1 *)))
1)))
(setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add j 1 n))
                      ((1 *))
                      work-%offset%)
(-
 (f2cl-lib:fref work-%data%
                ((f2cl-lib:int-add j 1 n))
                ((1 *))
                work-%offset%)
 (ddot (f2cl-lib:int-sub j ki 1)
        (f2cl-lib:array-slice t$
                                double-float
                                ((+ ki 1)
                                (f2cl-lib:int-add j
                                1))
                                ((1 ldt) (1 *)))
1
(f2cl-lib:array-slice work
                        double-float
                        ((+ ki 1 n))
                        ((1 *)))
1)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10 var-11 var-12 var-13 var-14
  var-15 var-16 var-17)
 (dlaln2 t 2 1 smin one
  (f2cl-lib:array-slice t$
                        double-float
                        (j j)
                        ((1 ldt) (1 *)))
  ldt one one
  (f2cl-lib:array-slice work
                        double-float
                        ((+ j n))
                        ((1 *)))
  n wr zero x 2 scale xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10
  var-11 var-12 var-13 var-14))
(setf scale var-15)
(setf xnorm var-16)
(setf ierr var-17))

```

```

      (if (/= scale one)
        (dscal
          (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
          scale
          (f2cl-lib:array-slice work
                                double-float
                                ((+ ki n))
                                ((1 *)))
          1))
      (setf (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add j n))
                          ((1 *))
                          work-%offset%))
        (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
      (setf (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add j 1 n))
                          ((1 *))
                          work-%offset%))
        (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
      (setf vmax
        (max
          (abs
            (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add j n))
                          ((1 *))
                          work-%offset%))
          (abs
            (f2cl-lib:fref work-%data%
                          ((f2cl-lib:int-add j 1 n))
                          ((1 *))
                          work-%offset%))
          vmax))
      (setf vcrit (/ bignum vmax)))
label170))

(cond
  ((not over)
   (dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
          (f2cl-lib:array-slice work
                                double-float
                                ((+ ki n))
                                ((1 *)))
          1
          (f2cl-lib:array-slice vl
                                double-float
                                (ki is)
                                ((1 ldvl) (1 *)))
          1)
  (setf ii
    (f2cl-lib:int-sub
     (f2cl-lib:int-add

```

```

(idamax
  (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
  (f2cl-lib:array-slice vl
    double-float
    (ki is)
    ((1 ldvl) (1 *)))
  1)
ki)
1))
(setf remax
  (/ one
    (abs
      (f2cl-lib:fref vl-%data%
        (ii is)
        ((1 ldvl) (1 *))
        vl-%offset%))))
(dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) remax
  (f2cl-lib:array-slice vl
    double-float
    (ki is)
    ((1 ldvl) (1 *)))
  1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add ki
      (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (setf (f2cl-lib:fref vl-%data%
    (k is)
    ((1 ldvl) (1 *))
    vl-%offset%)
    zero))))
(t
  (if (< ki n)
    (dgemv "N" n (f2cl-lib:int-sub n ki) one
      (f2cl-lib:array-slice vl
        double-float
        (1 (f2cl-lib:int-add ki 1))
        ((1 ldvl) (1 *)))
      ldvl
      (f2cl-lib:array-slice work
        double-float
        ((+ ki 1 n))
        ((1 *)))
      1
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add ki n))
        ((1 *))
        work-%offset%))

```

```

(f2cl-lib:array-slice vl
                        double-float
                        (1 ki)
                        ((1 ldvl) (1 *)))

1))
(setf ii
      (idamax n
              (f2cl-lib:array-slice vl
                                      double-float
                                      (1 ki)
                                      ((1 ldvl) (1 *)))

              1))
(setf remax
      (/ one
         (abs
          (f2cl-lib:fref vl-%data%
                          (ii ki)
                          ((1 ldvl) (1 *))
                          vl-%offset%))))
(dscal n remax
      (f2cl-lib:array-slice vl
                              double-float
                              (1 ki)
                              ((1 ldvl) (1 *)))

      1))))
(t
 (tagbody
  (cond
   ((>=
    (abs
     (f2cl-lib:fref t$
                     (ki (f2cl-lib:int-add ki 1))
                     ((1 ldt) (1 *))))

    (abs
     (f2cl-lib:fref t$
                     ((f2cl-lib:int-add ki 1) ki)
                     ((1 ldt) (1 *))))))
   (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add ki n))
                        ((1 *))
                        work-%offset%)

          (/ wi
             (f2cl-lib:fref t$-%data%
                             (ki (f2cl-lib:int-add ki 1))
                             ((1 ldt) (1 *))
                             t$-%offset%)))

   (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add ki 1 n2))
                        ((1 *))
                        work-%offset%)

```

```

                                one))
(t
  (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add ki n))
                        ((1 *))
                        work-%offset%))

                                one)
  (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add ki 1 n2))
                        ((1 *))
                        work-%offset%))

  (/ (- wi)
     (f2cl-lib:fref t$-%data%
                     ((f2cl-lib:int-add ki 1) ki)
                     ((1 ldt) (1 *))
                     t$-%offset%))))))
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add ki 1 n))
                    ((1 *))
                    work-%offset%))

                                zero)
(setf (f2cl-lib:fref work-%data%
                    ((f2cl-lib:int-add ki n2))
                    ((1 *))
                    work-%offset%))

                                zero)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 2)
              (f2cl-lib:int-add k 1))
              (> k n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add k n))
                      ((1 *))
                      work-%offset%))

  (*
    (-
      (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add ki n))
                      ((1 *))
                      work-%offset%))
      (f2cl-lib:fref t$-%data%
                      (ki k)
                      ((1 ldt) (1 *))
                      t$-%offset%))))
  (setf (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add k n2))
                      ((1 *))
                      work-%offset%))

  (*
    (-

```

```

(f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add ki 1 n2))
  ((1 *))
  work-%offset%))
(f2cl-lib:fref t$-%data%
  ((f2cl-lib:int-add ki 1) k)
  ((1 ldt) (1 *))
  t$-%offset%))))))
(setf vmax one)
(setf vcrit bignum)
(setf jnxt (f2cl-lib:int-add ki 2))
(f2cl-lib:fdo (j (f2cl-lib:int-add ki 2)
  (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (if (< j jnxt) (go label200))
  (setf j1 j)
  (setf j2 j)
  (setf jnxt (f2cl-lib:int-add j 1))
  (cond
    ((< j n)
     (cond
      (/=
        (f2cl-lib:fref t$
          ((f2cl-lib:int-add j 1) j)
          ((1 ldt) (1 *)))
        zero)
      (setf j2 (f2cl-lib:int-add j 1))
      (setf jnxt (f2cl-lib:int-add j 2))))))
  (cond
    (= j1 j2)
    (cond
      (> (f2cl-lib:fref work (j) ((1 *))) vcrit)
      (setf rec (/ one vmax))
      (dscal
        (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
        (f2cl-lib:array-slice work
          double-float
          ((+ ki n))
          ((1 *)))
        1)
      (dscal
        (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
        (f2cl-lib:array-slice work
          double-float
          ((+ ki n2))
          ((1 *)))
        1)
      (setf vmax one)
      (setf vcrit bignum))))

```



```

(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n))
                     ((1 *))
                     work-%offset%))
(-
  (f2cl-lib:fref work-%data%
                 ((f2cl-lib:int-add j n))
                 ((1 *))
                 work-%offset%))
  (ddot (f2cl-lib:int-sub j ki 2)
        (f2cl-lib:array-slice t$
                               double-float
                               ((+ ki 2) j)
                               ((1 ldt) (1 *)))
        1
        (f2cl-lib:array-slice work
                               double-float
                               ((+ ki 2 n))
                               ((1 *)))
        1)))
(setf (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j n2))
                     ((1 *))
                     work-%offset%))
(-
  (f2cl-lib:fref work-%data%
                 ((f2cl-lib:int-add j n2))
                 ((1 *))
                 work-%offset%))
  (ddot (f2cl-lib:int-sub j ki 2)
        (f2cl-lib:array-slice t$
                               double-float
                               ((+ ki 2) j)
                               ((1 ldt) (1 *)))
        1
        (f2cl-lib:array-slice work
                               double-float
                               ((+ ki 2 n2))
                               ((1 *)))
        1)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9 var-10 var-11 var-12 var-13
   var-14 var-15 var-16 var-17)
  (dlaln2 nil 1 2 smin one
          (f2cl-lib:array-slice t$
                                double-float
                                (j j)
                                ((1 ldt) (1 *)))
          ldt one one

```

```

(f2cl-lib:array-slice work
  double-float
  ((+ j n))
  ((1 *)))
  n wr (- wi) x 2 scale xnorm ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4
  var-5 var-6 var-7 var-8 var-9
  var-10 var-11 var-12 var-13
  var-14))
(setf scale var-15)
(setf xnorm var-16)
(setf ierr var-17))
(cond
  (/= scale one)
  (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
      double-float
      ((+ ki n))
      ((1 *)))
    1)
  (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
      double-float
      ((+ ki n2))
      ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n2))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(setf vmax
  (max
    (abs
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add j n))
        ((1 *))
        work-%offset%))
    (abs
      (f2cl-lib:fref work-%data%
        ((f2cl-lib:int-add j n2))
        ((1 *))
        work-%offset%)))))

```

```

                                ((1 *))
                                work-%offset%))
                                vmax))
(setf vcrit (/ bignum vmax)))
(t
  (setf beta
    (max
      (f2cl-lib:fref work-%data%
                     (j)
                     ((1 *))
                     work-%offset%))
      (f2cl-lib:fref work-%data%
                     ((f2cl-lib:int-add j 1))
                     ((1 *))
                     work-%offset%)))
    (cond
      ((> beta vcrit)
       (setf rec (/ one vmax))
       (dscal
        (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
        (f2cl-lib:array-slice work
                               double-float
                               ((+ ki n))
                               ((1 *)))
        1)
       (dscal
        (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1) rec
        (f2cl-lib:array-slice work
                               double-float
                               ((+ ki n2))
                               ((1 *)))
        1)
       (setf vmax one)
       (setf vcrit bignum)))
    (setf (f2cl-lib:fref work-%data%
                        ((f2cl-lib:int-add j n))
                        ((1 *))
                        work-%offset%))
      (-
        (f2cl-lib:fref work-%data%
                       ((f2cl-lib:int-add j n))
                       ((1 *))
                       work-%offset%))
        (ddot (f2cl-lib:int-sub j ki 2)
              (f2cl-lib:array-slice t$
                                     double-float
                                     ((+ ki 2) j)
                                     ((1 ldt) (1 *)))
              1
              (f2cl-lib:array-slice work

```

```

double-float
((+ ki 2 n))
((1 *)))

1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n2))
  ((1 *))
  work-%offset%)
(-
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add j n2))
    ((1 *))
    work-%offset%)
  (ddot (f2cl-lib:int-sub j ki 2)
    (f2cl-lib:array-slice t$
      double-float
      ((+ ki 2) j)
      ((1 ldt) (1 *)))
    1
    (f2cl-lib:array-slice work
      double-float
      ((+ ki 2 n2))
      ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j 1 n))
  ((1 *))
  work-%offset%)
(-
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add j 1 n))
    ((1 *))
    work-%offset%)
  (ddot (f2cl-lib:int-sub j ki 2)
    (f2cl-lib:array-slice t$
      double-float
      ((+ ki 2)
        (f2cl-lib:int-add j
          1))
      ((1 ldt) (1 *)))
    1
    (f2cl-lib:array-slice work
      double-float
      ((+ ki 2 n))
      ((1 *)))
    1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j 1 n2))
  ((1 *))
  work-%offset%)

```

```

(-
  (f2cl-lib:fref work-%data%
    ((f2cl-lib:int-add j 1 n2))
    ((1 *))
    work-%offset%)
  (ddot (f2cl-lib:int-sub j ki 2)
    (f2cl-lib:array-slice t$
      double-float
      ((+ ki 2)
        (f2cl-lib:int-add j
          1))
      ((1 ldt) (1 *)))
    1
    (f2cl-lib:array-slice work
      double-float
      ((+ ki 2 n2))
      ((1 *)))
    1)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9 var-10 var-11 var-12 var-13
   var-14 var-15 var-16 var-17)
  (dlaln2 t 2 2 smin one
    (f2cl-lib:array-slice t$
      double-float
      (j j)
      ((1 ldt) (1 *)))
    ldt one one
    (f2cl-lib:array-slice work
      double-float
      ((+ j n))
      ((1 *)))
    n wr (- wi) x 2 scale xnorm ierr)
  (declare (ignore var-0 var-1 var-2 var-3 var-4
    var-5 var-6 var-7 var-8 var-9
    var-10 var-11 var-12 var-13
    var-14))
  (setf scale var-15)
  (setf xnorm var-16)
  (setf ierr var-17))
(cond
  ((/= scale one)
   (dscal
    (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
    scale
    (f2cl-lib:array-slice work
      double-float
      ((+ ki n))
      ((1 *)))
    1)

```

```

(dscal
  (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
  scale
  (f2cl-lib:array-slice work
    double-float
    ((+ ki n2))
    ((1 *)))
  1)))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j n2))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j 1 n))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
(setf (f2cl-lib:fref work-%data%
  ((f2cl-lib:int-add j 1 n2))
  ((1 *))
  work-%offset%)
  (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
(setf vmax
  (max
    (abs (f2cl-lib:fref x (1 1) ((1 2) (1 2))))
    (abs (f2cl-lib:fref x (1 2) ((1 2) (1 2))))
    (abs (f2cl-lib:fref x (2 1) ((1 2) (1 2))))
    (abs (f2cl-lib:fref x (2 2) ((1 2) (1 2))))
    vmax))
  (setf vcrit (/ bignum vmax))))

label200))
label210

(cond
  ((not over)
    (dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
      (f2cl-lib:array-slice work
        double-float
        ((+ ki n))
        ((1 *)))
      1
      (f2cl-lib:array-slice vl
        double-float
        (ki is)
        ((1 ldvl) (1 *)))

```

```

1)
(dcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
 (f2cl-lib:array-slice work
                          double-float
                          ((+ ki n2))
                          ((1 *)))
1
(f2cl-lib:array-slice vl
                      double-float
                      (ki (f2cl-lib:int-add is 1))
                      ((1 ldvl) (1 *)))
1)
(setf emax zero)
(f2cl-lib:fdo (k ki (f2cl-lib:int-add k 1))
              (> k n) nil)
  (tagbody
    (setf emax
          (max emax
                (+
                 (abs
                  (f2cl-lib:fref vl-%data%
                                (k is)
                                ((1 ldvl) (1 *))
                                vl-%offset%))
                 (abs
                  (f2cl-lib:fref vl-%data%
                                (k
                                 (f2cl-lib:int-add is
                                   1))
                                ((1 ldvl) (1 *))
                                vl-%offset%)))))))
  (setf remax (/ one emax))
(dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
 remax
 (f2cl-lib:array-slice vl
                      double-float
                      (ki is)
                      ((1 ldvl) (1 *)))
1)
(dscal (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
 remax
 (f2cl-lib:array-slice vl
                      double-float
                      (ki (f2cl-lib:int-add is 1))
                      ((1 ldvl) (1 *)))
1)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              (> k
                (f2cl-lib:int-add ki
                                   (f2cl-lib:int-sub

```

```

1)))
nil)
(tagbody
  (setf (f2cl-lib:fref vl-%data%
                      (k is)
                      ((1 ldvl) (1 *)))
        vl-%offset%)
    zero)
  (setf (f2cl-lib:fref vl-%data%
                      (k (f2cl-lib:int-add is 1))
                      ((1 ldvl) (1 *)))
        vl-%offset%)
    zero))))
(t
  (cond
    ((< ki (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
      (dgemv "N" n (f2cl-lib:int-sub n ki 1) one
        (f2cl-lib:array-slice vl
                              double-float
                              (1 (f2cl-lib:int-add ki 2))
                              ((1 ldvl) (1 *)))
        ldvl
        (f2cl-lib:array-slice work
                              double-float
                              ((+ ki 2 n))
                              ((1 *)))
        1
        (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add ki n))
                      ((1 *))
                      work-%offset%)
        (f2cl-lib:array-slice vl
                              double-float
                              (1 ki)
                              ((1 ldvl) (1 *)))
        1)
      (dgemv "N" n (f2cl-lib:int-sub n ki 1) one
        (f2cl-lib:array-slice vl
                              double-float
                              (1 (f2cl-lib:int-add ki 2))
                              ((1 ldvl) (1 *)))
        ldvl
        (f2cl-lib:array-slice work
                              double-float
                              ((+ ki 2 n2))
                              ((1 *)))
        1
        (f2cl-lib:fref work-%data%
                      ((f2cl-lib:int-add ki 1 n2))
                      ((1 *))

```



```

                                work-%offset%)
(f2cl-lib:array-slice vl
                        double-float
                        (1 (f2cl-lib:int-add ki 1))
                        ((1 ldvl) (1 *)))
1))
(t
(dscal n
(f2cl-lib:fref work-%data%
                ((f2cl-lib:int-add ki n))
                ((1 *))
                work-%offset%)
(f2cl-lib:array-slice vl
                        double-float
                        (1 ki)
                        ((1 ldvl) (1 *)))
1)
(dscal n
(f2cl-lib:fref work-%data%
                ((f2cl-lib:int-add ki 1 n2))
                ((1 *))
                work-%offset%)
(f2cl-lib:array-slice vl
                        double-float
                        (1 (f2cl-lib:int-add ki 1))
                        ((1 ldvl) (1 *)))
1)))
(setf emax zero)
(f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              ((> k n) nil)
(tagbody
 (setf emax
       (max emax
            (+
             (abs
              (f2cl-lib:fref vl-%data%
                              (k ki)
                              ((1 ldvl) (1 *))
                              vl-%offset%))
             (abs
              (f2cl-lib:fref vl-%data%
                              (k
                               (f2cl-lib:int-add ki
                               1))
                              ((1 ldvl) (1 *))
                              vl-%offset%)))))))
(setf remax (/ one emax))
(dscal n remax
(f2cl-lib:array-slice vl
                        double-float

```

```

                                (1 ki)
                                ((1 ldvl) (1 *)))
                                1)
                                (dscal n remax
                                (f2cl-lib:array-slice vl
                                double-float
                                (1 (f2cl-lib:int-add ki 1))
                                ((1 ldvl) (1 *)))
                                1))))))
                                (setf is (f2cl-lib:int-add is 1))
                                (if (/= ip 0) (setf is (f2cl-lib:int-add is 1)))
label250
                                (if (= ip -1) (setf ip 0))
                                (if (= ip 1) (setf ip -1))))))
end_label
                                (return
                                (values nil nil nil nil nil nil nil nil nil nil nil m nil info))))))

```

dtrexc LAPACK

— dtrexc.input —

```

)set break resume
)sys rm -f dtrexc.output
)spool dtrexc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— dtrexc.help —

```

=====
dtrexc examples
=====

=====
Man Page Details
=====

```

NAME

DTREXC - the real Schur factorization of a real matrix $A = Q^*TQ$, so that the diagonal block of T with row index IFST is moved to row ILST

SYNOPSIS

SUBROUTINE DTREXC(COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK, INFO)

CHARACTER COMPQ

INTEGER IFST, ILST, INFO, LDQ, LDT, N

DOUBLE PRECISION Q(LDQ, *), T(LDT, *), WORK(*)

Purpose

=====

DTREXC reorders the real Schur factorization of a real matrix $A = Q^*TQ$, so that the diagonal block of T with row index IFST is moved to row ILST.

The real Schur form T is reordered by an orthogonal similarity transformation Z^*T^*Z , and optionally the matrix Q of Schur vectors is updated by postmultiplying it with Z .

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

=====

COMPQ (input) CHARACTER*1
 = 'V': update the matrix Q of Schur vectors;
 = 'N': do not update Q .

N (input) INTEGER
 The order of the matrix T . $N \geq 0$.

T (input/output) DOUBLE PRECISION array, dimension (LDT,N)
 On entry, the upper quasi-triangular matrix T , in Schur canonical form.
 On exit, the reordered upper quasi-triangular matrix, again in Schur canonical form.

LDT (input) INTEGER
 The leading dimension of the array T . $LDT \geq \max(1,N)$.

Q (input/output) DOUBLE PRECISION array, dimension (LDQ,N)

On entry, if COMPQ = 'V', the matrix Q of Schur vectors.
 On exit, if COMPQ = 'V', Q has been postmultiplied by the
 orthogonal transformation matrix Z which reorders T.
 If COMPQ = 'N', Q is not referenced.

LDQ (input) INTEGER
 The leading dimension of the array Q. LDQ \geq max(1,N).

IFST (input/output) INTEGER
 ILST (input/output) INTEGER
 Specify the reordering of the diagonal blocks of T.
 The block with row index IFST is moved to row ILST, by a
 sequence of transpositions between adjacent blocks.
 On exit, if IFST pointed on entry to the second row of a
 2-by-2 block, it is changed to point to the first row; ILST
 always points to the first row of the block in its final
 position (which may differ from its input value by +1 or -1).
 1 \leq IFST \leq N; 1 \leq ILST \leq N.

WORK (workspace) DOUBLE PRECISION array, dimension (N)

INFO (output) INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value
 = 1: two adjacent blocks were too close to swap (the problem
 is very ill-conditioned); T may have been partially
 reordered, and ILST points to the first row of the
 current position of the block being moved.

— dtrexc.f —

```

SUBROUTINE DTREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK,
$                INFO )
*
*  -- LAPACK routine (version 3.0) --
*    Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*    Courant Institute, Argonne National Lab, and Rice University
*    March 31, 1993
*
*    .. Scalar Arguments ..
CHARACTER      COMPQ
INTEGER        IFST, ILST, INFO, LDQ, LDT, N
*
*    .. Array Arguments ..
DOUBLE PRECISION Q( LDQ, * ), T( LDT, * ), WORK( * )
*

```

```

*
* =====
*
* .. Parameters ..
DOUBLE PRECISION    ZERO
PARAMETER            ( ZERO = 0.0D+0 )
*
* ..
* .. Local Scalars ..
LOGICAL              WANTQ
INTEGER              HERE, NBF, NBL, NBNEXT
*
* ..
* .. External Functions ..
LOGICAL              LSAME
EXTERNAL             LSAME
*
* ..
* .. External Subroutines ..
EXTERNAL             DLAEXC, XERBLA
*
* ..
* .. Intrinsic Functions ..
INTRINSIC            MAX
*
* ..
* .. Executable Statements ..
*
* Decode and test the input arguments.
*
*
* INFO = 0
* WANTQ = LSAME( COMPQ, 'V' )
* IF( .NOT.WANTQ .AND. .NOT.LSAME( COMPQ, 'N' ) ) THEN
*   INFO = -1
* ELSE IF( N.LT.0 ) THEN
*   INFO = -2
* ELSE IF( LDT.LT.MAX( 1, N ) ) THEN
*   INFO = -4
* ELSE IF( LDQ.LT.1 .OR. ( WANTQ .AND. LDQ.LT.MAX( 1, N ) ) ) THEN
*   INFO = -6
* ELSE IF( IFST.LT.1 .OR. IFST.GT.N ) THEN
*   INFO = -7
* ELSE IF( ILST.LT.1 .OR. ILST.GT.N ) THEN
*   INFO = -8
* END IF
* IF( INFO.NE.0 ) THEN
*   CALL XERBLA( 'DTREXC', -INFO )
*   RETURN
* END IF
*
* Quick return if possible
*
* IF( N.LE.1 )
$  RETURN
*

```

```

*      Determine the first row of specified block
*      and find out it is 1 by 1 or 2 by 2.
*
      IF( IFST.GT.1 ) THEN
        IF( T( IFST, IFST-1 ).NE.ZERO )
$          IFST = IFST - 1
      END IF
      NBF = 1
      IF( IFST.LT.N ) THEN
        IF( T( IFST+1, IFST ).NE.ZERO )
$          NBF = 2
      END IF

*
*      Determine the first row of the final block
*      and find out it is 1 by 1 or 2 by 2.
*
      IF( ILST.GT.1 ) THEN
        IF( T( ILST, ILST-1 ).NE.ZERO )
$          ILST = ILST - 1
      END IF
      NBL = 1
      IF( ILST.LT.N ) THEN
        IF( T( ILST+1, ILST ).NE.ZERO )
$          NBL = 2
      END IF

*
      IF( IFST.EQ.ILST )
$        RETURN
      *
      IF( IFST.LT.ILST ) THEN
*
*      Update ILST
*
        IF( NBF.EQ.2 .AND. NBL.EQ.1 )
$          ILST = ILST - 1
        IF( NBF.EQ.1 .AND. NBL.EQ.2 )
$          ILST = ILST + 1
*
        HERE = IFST
*
10      CONTINUE
*
*      Swap block with next one below
*
      IF( NBF.EQ.1 .OR. NBF.EQ.2 ) THEN
*
*      Current block either 1 by 1 or 2 by 2
*
        NBNEXT = 1
        IF( HERE+NBF+1.LE.N ) THEN

```

```

                IF( T( HERE+NBF+1, HERE+NBF ).NE.ZERO )
$              NBNEXT = 2
            END IF
            CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, NBF, NBNEXT,
$              WORK, INFO )
            IF( INFO.NE.0 ) THEN
                ILST = HERE
                RETURN
            END IF
            HERE = HERE + NBNEXT
*
*      Test if 2 by 2 block breaks into two 1 by 1 blocks
*
            IF( NBF.EQ.2 ) THEN
                IF( T( HERE+1, HERE ).EQ.ZERO )
$              NBF = 3
            END IF
*
            ELSE
*
*      Current block consists of two 1 by 1 blocks each of which
*      must be swapped individually
*
            NBNEXT = 1
            IF( HERE+3.LE.N ) THEN
                IF( T( HERE+3, HERE+2 ).NE.ZERO )
$              NBNEXT = 2
            END IF
            CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE+1, 1, NBNEXT,
$              WORK, INFO )
            IF( INFO.NE.0 ) THEN
                ILST = HERE
                RETURN
            END IF
            IF( NBNEXT.EQ.1 ) THEN
*
*      Swap two 1 by 1 blocks, no problems possible
*
                CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, 1, NBNEXT,
$              WORK, INFO )
                HERE = HERE + 1
            ELSE
*
*      Recompute NBNEXT in case 2 by 2 split
*
                IF( T( HERE+2, HERE+1 ).EQ.ZERO )
$              NBNEXT = 1
                IF( NBNEXT.EQ.2 ) THEN
*
*      2 by 2 Block did not split

```

```

*
*      CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, 1,
$              NBNEXT, WORK, INFO )
*      IF( INFO.NE.0 ) THEN
*          ILST = HERE
*          RETURN
*      END IF
*      HERE = HERE + 2
*      ELSE
*
*          2 by 2 Block did split
*
*      CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, 1, 1,
$              WORK, INFO )
*      CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE+1, 1, 1,
$              WORK, INFO )
*      HERE = HERE + 2
*      END IF
*      END IF
*      END IF
*      IF( HERE.LT.ILST )
$          GO TO 10
*
*      ELSE
*
*          HERE = IFST
20      CONTINUE
*
*      Swap block with next one above
*
*      IF( NBF.EQ.1 .OR. NBF.EQ.2 ) THEN
*
*          Current block either 1 by 1 or 2 by 2
*
*          NBNEXT = 1
*          IF( HERE.GE.3 ) THEN
*              IF( T( HERE-1, HERE-2 ).NE.ZERO )
$                  NBNEXT = 2
*          END IF
*          CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE-NBNEXT, NBNEXT,
$              NBF, WORK, INFO )
*          IF( INFO.NE.0 ) THEN
*              ILST = HERE
*              RETURN
*          END IF
*          HERE = HERE - NBNEXT
*
*      Test if 2 by 2 block breaks into two 1 by 1 blocks
*
*      IF( NBF.EQ.2 ) THEN

```



```

                IF( T( HERE+1, HERE ).EQ.ZERO )
$              NBF = 3
            END IF
*
        ELSE
*
*          Current block consists of two 1 by 1 blocks each of which
*          must be swapped individually
*
            NBNEXT = 1
            IF( HERE.GE.3 ) THEN
                IF( T( HERE-1, HERE-2 ).NE.ZERO )
$              NBNEXT = 2
            END IF
            CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE-NBNEXT, NBNEXT,
$              1, WORK, INFO )
            IF( INFO.NE.0 ) THEN
                ILST = HERE
                RETURN
            END IF
            IF( NBNEXT.EQ.1 ) THEN
*
*              Swap two 1 by 1 blocks, no problems possible
*
                CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, NBNEXT, 1,
$              WORK, INFO )
                HERE = HERE - 1
            ELSE
*
*              Recompute NBNEXT in case 2 by 2 split
*
                IF( T( HERE, HERE-1 ).EQ.ZERO )
$              NBNEXT = 1
                IF( NBNEXT.EQ.2 ) THEN
*
*              2 by 2 Block did not split
*
                    CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE-1, 2, 1,
$              WORK, INFO )
                    IF( INFO.NE.0 ) THEN
                        ILST = HERE
                        RETURN
                    END IF
                    HERE = HERE - 2
                ELSE
*
*              2 by 2 Block did split
*
                    CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE, 1, 1,
$              WORK, INFO )

```

```

                CALL DLAEXC( WANTQ, N, T, LDT, Q, LDQ, HERE-1, 1, 1,
$                WORK, INFO )
                HERE = HERE - 2
            END IF
        END IF
    END IF
    IF( HERE.GT.ILST )
$        GO TO 20
    END IF
    ILST = HERE
*
*   RETURN
*
*   End of DTREXC
*
END

```

— LAPACK dtrexc —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun dtrexc (compq n t$ ldt q ldq ifst ilst work info)
    (declare (type (simple-array double-float (*)) work q t$)
              (type fixnum info ilst ifst ldq ldt n)
              (type character compq))
    (f2cl-lib:with-multi-array-data
      ((compq character compq-%data% compq-%offset%)
       (t$ double-float t$-%data% t$-%offset%)
       (q double-float q-%data% q-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((here 0) (nbf 0) (nbl 0) (nbnext 0) (wantq nil))
        (declare (type fixnum here nbf nbl nbnext)
                  (type (member t nil) wantq))
        (setf info 0)
        (setf wantq (char-equal compq #\V))
        (cond
          ((and (not wantq) (not (char-equal compq #\N)))
           (setf info -1))
          ((< n 0)
           (setf info -2))
          ((< ldt (max (the fixnum 1) (the fixnum n)))
           (setf info -4))
          ((or (< ldq 1)
               (and wantq
                    (< ldq
                     (max (the fixnum 1)

```

```

                                (the fixnum n))))
    (setf info -6))
  ((or (< ifst 1) (> ifst n))
   (setf info -7))
  ((or (< ilst 1) (> ilst n))
   (setf info -8)))
(cond
  (/= info 0)
  (error
   " ** On entry to ~a parameter number ~a had an illegal value~%"
   "DTREXC" (f2cl-lib:int-sub info))
  (go end_label)))
(if (<= n 1) (go end_label))
(cond
  (> ifst 1)
  (if
   (/=
    (f2cl-lib:fref t$-%data%
                   (ifst (f2cl-lib:int-sub ifst 1))
                   ((1 ldt) (1 *)))
    t$-%offset%)
   zero)
  (setf ifst (f2cl-lib:int-sub ifst 1))))
(setf nbf 1)
(cond
  (< ifst n)
  (if
   (/=
    (f2cl-lib:fref t$-%data%
                   ((f2cl-lib:int-add ifst 1) ifst)
                   ((1 ldt) (1 *)))
    t$-%offset%)
   zero)
  (setf nbf 2))))
(cond
  (> ilst 1)
  (if
   (/=
    (f2cl-lib:fref t$-%data%
                   (ilst (f2cl-lib:int-sub ilst 1))
                   ((1 ldt) (1 *)))
    t$-%offset%)
   zero)
  (setf ilst (f2cl-lib:int-sub ilst 1))))
(setf nbl 1)
(cond
  (< ilst n)
  (if
   (/=
    (f2cl-lib:fref t$-%data%

```

```

                                ((f2cl-lib:int-add ilst 1) ilst)
                                ((1 ldt) (1 *))
                                t$-%offset%)

                                zero)
                                (setf nbl 2))))
(if (= ifst ilst) (go end_label))
(cond
  ((< ifst ilst)
    (tagbody
      (if (and (= nbf 2) (= nbl 1))
          (setf ilst (f2cl-lib:int-sub ilst 1)))
      (if (and (= nbf 1) (= nbl 2))
          (setf ilst (f2cl-lib:int-add ilst 1)))
      (setf here ifst)

label10
      (cond
        ((or (= nbf 1) (= nbf 2))
          (setf nbnext 1)
          (cond
            ((<= (f2cl-lib:int-add here nbf 1) n)
              (if
                (/=
                  (f2cl-lib:fref t$-%data%
                                ((f2cl-lib:int-add here nbf 1)
                                (f2cl-lib:int-add here nbf))
                                ((1 ldt) (1 *)))
                  t$-%offset%)

                zero)
              (setf nbnext 2))))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
               var-9 var-10)
              (dlaexc wantq n t$ ldt q ldq here nbf nbnext work info)
              (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
                               var-7 var-8 var-9))
              (setf info var-10))
              (cond
                ((/= info 0)
                  (setf ilst here)
                  (go end_label)))
              (setf here (f2cl-lib:int-add here nbnext))
              (cond
                ((= nbf 2)
                  (if
                    (=
                      (f2cl-lib:fref t$-%data%
                                    ((f2cl-lib:int-add here 1) here)
                                    ((1 ldt) (1 *)))
                      t$-%offset%)

                    zero)
                  (setf nbnext 2))))
                (t$-%offset%)

                zero)
                (setf nbnext 2))))
  ))

```

```

        (setf nbf 3))))))
(t
 (setf nbnext 1)
 (cond
  ((<= (f2cl-lib:int-add here 3) n)
   (if
    (/=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add here 3)
                     (f2cl-lib:int-add here 2))
                    ((1 ldt) (1 *)))
     t$-%offset%)
    zero)
   (setf nbnext 2))))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10)
  (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-add here 1) 1
   nbnext work info)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9))
  (setf info var-10))
 (cond
  ((/= info 0)
   (setf ilst here)
   (go end_label)))
 (cond
  ((= nbnext 1)
   (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10)
    (dlaexc wantq n t$ ldt q ldq here 1 nbnext work info)
    (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
     var-7 var-8 var-9))
    (setf info var-10))
   (setf here (f2cl-lib:int-add here 1)))
  (t
   (if
    (=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add here 2)
                     (f2cl-lib:int-add here 1))
                    ((1 ldt) (1 *)))
     t$-%offset%)
    zero)
   (setf nbnext 1))
  (cond
   ((= nbnext 2)
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```

```

        var-8 var-9 var-10)
      (dlaexc wantq n t$ ldt q ldq here 1 nbnext work info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                    var-6 var-7 var-8 var-9))
      (setf info var-10))
    (cond
      ((/= info 0)
       (setf ilst here)
       (go end_label)))
    (setf here (f2cl-lib:int-add here 2)))
  (t
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10)
     (dlaexc wantq n t$ ldt q ldq here 1 1 work info)
     (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9))
     (setf info var-10))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
      (dlaexc wantq n t$ ldt q ldq
        (f2cl-lib:int-add here 1) 1 1 work info)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                      var-6 var-7 var-8 var-9))
      (setf info var-10))
      (setf here (f2cl-lib:int-add here 2))))))
    (if (< here ilst) (go label10)))
  (t
   (tagbody
    (setf here ifst)
label120
    (cond
      ((or (= nbf 1) (= nbf 2))
       (setf nbnext 1)
       (cond
         ((>= here 3)
          (if
           (/=
            (f2cl-lib:fref t$-%data%
                          ((f2cl-lib:int-sub here 1)
                           (f2cl-lib:int-sub here 2))
                          ((1 ldt) (1 *)))
             t$-%offset%)
           zero)
          (setf nbnext 2))))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
         var-9 var-10)
        (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-sub here nbnext)

```

```

        nbnext nbf work info)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9))
(setf info var-10))
(cond
  ((/= info 0)
   (setf ilst here)
   (go end_label)))
(setf here (f2cl-lib:int-sub here nbnext))
(cond
  ((= nbf 2)
   (if
    (=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-add here 1) here)
                    ((1 ldt) (1 *)))
     t$-%offset%)
    zero)
   (setf nbf 3))))))
(t
 (setf nbnext 1)
 (cond
  ((>= here 3)
   (if
    (/=
     (f2cl-lib:fref t$-%data%
                    ((f2cl-lib:int-sub here 1)
                     (f2cl-lib:int-sub here 2))
                    ((1 ldt) (1 *)))
     t$-%offset%)
    zero)
   (setf nbnext 2))))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10)
 (dlaexc wantq n t$ ldt q ldq (f2cl-lib:int-sub here nbnext)
  nbnext 1 work info)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9))
 (setf info var-10))
 (cond
  ((/= info 0)
   (setf ilst here)
   (go end_label)))
 (cond
  ((= nbnext 1)
   (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10)
    (dlaexc wantq n t$ ldt q ldq here nbnext 1 work info)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8 var-9))
(setf info var-10))
(setf here (f2cl-lib:int-sub here 1)))
(t
  (if
    (=
      (f2cl-lib:fref t$-%data%
                     (here (f2cl-lib:int-sub here 1))
                     ((1 ldt) (1 *))
                     t$-%offset%)

      zero)
    (setf nbnext 1))
  (cond
    ((= nbnext 2)
     (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
       (dlaexc wantq n t$ ldt q ldq
                (f2cl-lib:int-sub here 1) 2 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
     (cond
       ((/= info 0)
        (setf ilst here)
        (go end_label)))
     (setf here (f2cl-lib:int-sub here 2)))
    (t
     (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
       (dlaexc wantq n t$ ldt q ldq here 1 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
     (multiple-value-bind
       (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
       (dlaexc wantq n t$ ldt q ldq
                (f2cl-lib:int-sub here 1) 1 1 work info)
       (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
                        var-6 var-7 var-8 var-9))
       (setf info var-10))
     (setf here (f2cl-lib:int-sub here 2))))))
  (if (> here ilst) (go label20))))
(setf ilst here)
end_label
(return (values nil nil nil nil nil nil ifst ilst nil info))))

```

dtrsna LAPACK

— dtrsna.input —

```
)set break resume
)sys rm -f dtrsna.output
)spool dtrsna.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— dtrsna.help —

```
=====
dtrsna examples
=====
```

```
=====
Man Page Details
=====
```

NAME

DTRSNA - reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T (or of any matrix Q^*TQ with Q orthogonal)

SYNOPSIS

```
SUBROUTINE DTRSNA( JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR, LDVR,
                  S, SEP, MM, M, WORK, LDWORK, IWORK, INFO )
```

CHARACTER HOWMNY, JOB

INTEGER INFO, LDT, LDVL, LDVR, LDWORK, M, MM, N

LOGICAL SELECT(*)

INTEGER IWORK(*)

DOUBLE PRECISION S(*), SEP(*), T(LDT, *), VL(LDVL, *

```
), VR( LDVR, * ), WORK( LDWORK, * )
```

Purpose

```
=====
```

DTRSNA estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T (or of any matrix Q^*TQ with Q orthogonal).

T must be in Schur canonical form (as returned by DHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

```
=====
```

JOB (input) CHARACTER*1
Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (SEP):
= 'E': for eigenvalues only (S);
= 'V': for eigenvectors only (SEP);
= 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY (input) CHARACTER*1
= 'A': compute condition numbers for all eigenpairs;
= 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT (input) LOGICAL array, dimension (N)
If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue $w(j)$, SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues $w(j)$ and $w(j+1)$, either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE..
If HOWMNY = 'A', SELECT is not referenced.

N (input) INTEGER
The order of the matrix T. $N \geq 0$.

T (input) DOUBLE PRECISION array, dimension (LDT,N)
The upper quasi-triangular matrix T, in Schur canonical form.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq \max(1,N)$.

VL (input) DOUBLE PRECISION array, dimension (LDVL,M)

If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any Q^*TQ with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by DHSEIN or DTREVC.
 If JOB = 'V', VL is not referenced.

- LDVL (input) INTEGER
 The leading dimension of the array VL.
 LDVL ≥ 1 ; and if JOB = 'E' or 'B', LDVL $\geq N$.
- VR (input) DOUBLE PRECISION array, dimension (LDVR,M)
 If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any Q^*TQ with Q orthogonal), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by DHSEIN or DTREVC.
 If JOB = 'V', VR is not referenced.
- LDVR (input) INTEGER
 The leading dimension of the array VR.
 LDVR ≥ 1 ; and if JOB = 'E' or 'B', LDVR $\geq N$.
- S (output) DOUBLE PRECISION array, dimension (MM)
 If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), SEP(j), and the j-th columns of VL and VR all correspond to the same eigenpair (but not in general the j-th eigenpair, unless all eigenpairs are selected).
 If JOB = 'V', S is not referenced.
- SEP (output) DOUBLE PRECISION array, dimension (MM)
 If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of SEP are set to the same value. If the eigenvalues cannot be reordered to compute SEP(j), SEP(j) is set to 0; this can only occur when the true value would be very small anyway.
 If JOB = 'E', SEP is not referenced.
- MM (input) INTEGER
 The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM $\geq M$.
- M (output) INTEGER
 The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers.

If HOWMNY = 'A', M is set to N.

WORK (workspace) DOUBLE PRECISION array, dimension (LDWORK,N+1)
If JOB = 'E', WORK is not referenced.

LDWORK (input) INTEGER
The leading dimension of the array WORK.
LDWORK >= 1; and if JOB = 'V' or 'B', LDWORK >= N.

IWORK (workspace) INTEGER array, dimension (N)
If JOB = 'E', IWORK is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

Further Details

=====

The reciprocal of the condition number of an eigenvalue lambda is defined as

$$S(\text{lambda}) = |v' * u| / (\text{norm}(u) * \text{norm}(v))$$

where u and v are the right and left eigenvectors of T corresponding to lambda; v' denotes the conjugate-transpose of v, and norm(u) denotes the Euclidean norm. These reciprocal condition numbers always lie between zero (very badly conditioned) and one (very well conditioned). If n = 1, S(lambda) is defined to be 1.

An approximate error bound for a computed eigenvalue W(i) is given by

$$\text{EPS} * \text{norm}(T) / S(i)$$

where EPS is the machine precision.

The reciprocal of the condition number of the right eigenvector u corresponding to lambda is defined as follows. Suppose

$$T = \begin{pmatrix} \text{lambda} & c \\ 0 & T22 \end{pmatrix}$$

Then the reciprocal condition number is

$$\text{SEP}(\text{lambda}, T22) = \text{sigma-min}(T22 - \text{lambda} * I)$$

where sigma-min denotes the smallest singular value. We approximate the smallest singular value by the reciprocal of an estimate of the one-norm of the inverse of T22 - lambda*I. If n = 1, SEP(1) is defined to be abs(T(1,1)).

An approximate error bound for a computed right eigenvector $VR(i)$ is given by

$$EPS * \text{norm}(T) / \text{SEP}(i)$$

— dtrsna.f —

```

SUBROUTINE DTRSNA( JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
$                LDVR, S, SEP, MM, M, WORK, LDWORK, IWORK,
$                INFO )
*
*  -- LAPACK routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  September 30, 1994
*
*  .. Scalar Arguments ..
CHARACTER          HOWMNY, JOB
INTEGER            INFO, LDT, LDVL, LDVR, LDWORK, M, MM, N
*
*  .. Array Arguments ..
LOGICAL            SELECT( * )
INTEGER            IWORK( * )
DOUBLE PRECISION   S( * ), SEP( * ), T( LDT, * ), VL( LDVL, * ),
$                VR( LDVR, * ), WORK( LDWORK, * )
*
*  ..
*
*  =====
*
*  .. Parameters ..
DOUBLE PRECISION   ZERO, ONE, TWO
PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0, TWO = 2.0D+0 )
*
*  ..
*  .. Local Scalars ..
LOGICAL            PAIR, SOMCON, WANTBH, WANTS, WANTSP
INTEGER            I, IERR, IFST, ILST, J, K, KASE, KS, N2, NN
DOUBLE PRECISION   BIGNUM, COND, CS, DELTA, DUMM, EPS, EST, LNRN,
$                MU, PROD, PROD1, PROD2, RNRN, SCALE, SMLNUM, SN
*
*  ..
*  .. Local Arrays ..
DOUBLE PRECISION   DUMMY( 1 )
*
*  ..
*  .. External Functions ..
LOGICAL            LSAME
DOUBLE PRECISION   DDOT, DLAMCH, DLAPY2, DNRN2

```



```

$           M = M + 2
           END IF
           ELSE
           IF( SELECT( N ) )
$           M = M + 1
           END IF
           END IF
10      CONTINUE
      ELSE
        M = N
      END IF
*
      IF( MM.LT.M ) THEN
        INFO = -13
      ELSE IF( LDWORK.LT.1 .OR. ( WANTSP .AND. LDWORK.LT.N ) ) THEN
        INFO = -16
      END IF
    END IF
    IF( INFO.NE.0 ) THEN
      CALL XERBLA( 'DTRSNA', -INFO )
      RETURN
    END IF
*
*   Quick return if possible
*
      IF( N.EQ.0 )
$      RETURN
*
      IF( N.EQ.1 ) THEN
        IF( SOMCON ) THEN
          IF( .NOT.SELECT( 1 ) )
$          RETURN
        END IF
        IF( WANTS )
$          S( 1 ) = ONE
        IF( WANTSP )
$          SEP( 1 ) = ABS( T( 1, 1 ) )
        RETURN
      END IF
*
*   Get machine constants
*
      EPS = DLAMCH( 'P' )
      SMLNUM = DLAMCH( 'S' ) / EPS
      BIGNUM = ONE / SMLNUM
      CALL DLABAD( SMLNUM, BIGNUM )
*
      KS = 0
      PAIR = .FALSE.
      DO 60 K = 1, N

```

```

*
*      Determine whether T(k,k) begins a 1-by-1 or 2-by-2 block.
*
      IF( PAIR ) THEN
        PAIR = .FALSE.
        GO TO 60
      ELSE
        IF( K.LT.N )
$          PAIR = T( K+1, K ).NE.ZERO
      END IF

*
*      Determine whether condition numbers are required for the k-th
*      eigenpair.
*
      IF( SOMCON ) THEN
        IF( PAIR ) THEN
          IF( .NOT.SELECT( K ) .AND. .NOT.SELECT( K+1 ) )
$            GO TO 60
          ELSE
            IF( .NOT.SELECT( K ) )
$              GO TO 60
          END IF
        END IF

*
      KS = KS + 1

*
      IF( WANTS ) THEN

*
*      Compute the reciprocal condition number of the k-th
*      eigenvalue.
*
      IF( .NOT.PAIR ) THEN

*
*      Real eigenvalue.
*
        PROD = DDOT( N, VR( 1, KS ), 1, VL( 1, KS ), 1 )
        RNRM = DNRM2( N, VR( 1, KS ), 1 )
        LNRM = DNRM2( N, VL( 1, KS ), 1 )
        S( KS ) = ABS( PROD ) / ( RNRM*LNRM )
      ELSE

*
*      Complex eigenvalue.
*
        PROD1 = DDOT( N, VR( 1, KS ), 1, VL( 1, KS ), 1 )
        PROD1 = PROD1 + DDOT( N, VR( 1, KS+1 ), 1, VL( 1, KS+1 ),
$          1 )
        PROD2 = DDOT( N, VL( 1, KS ), 1, VR( 1, KS+1 ), 1 )
        PROD2 = PROD2 - DDOT( N, VL( 1, KS+1 ), 1, VR( 1, KS ),
$          1 )
        RNRM = DLAPY2( DNRM2( N, VR( 1, KS ), 1 ),

```



```

$          DNRM2( N, VR( 1, KS+1 ), 1 ) )
          LNRM = DLAPY2( DNRM2( N, VL( 1, KS ), 1 ),
$          DNRM2( N, VL( 1, KS+1 ), 1 ) )
          COND = DLAPY2( PROD1, PROD2 ) / ( RNRM*LNRM )
          S( KS ) = COND
          S( KS+1 ) = COND
        END IF
      END IF

*
      IF( WANTSP ) THEN
*
*         Estimate the reciprocal condition number of the k-th
*         eigenvector.
*
*         Copy the matrix T to the array WORK and swap the diagonal
*         block beginning at T(k,k) to the (1,1) position.
*
          CALL DLACPY( 'Full', N, N, T, LDT, WORK, LDWORK )
          IFST = K
          ILST = 1
          CALL DTREXC( 'No Q', N, WORK, LDWORK, DUMMY, 1, IFST, ILST,
$          WORK( 1, N+1 ), IERR )
*
          IF( IERR.EQ.1 .OR. IERR.EQ.2 ) THEN
*
*             Could not swap because blocks not well separated
*
*             SCALE = ONE
*             EST = BIGNUM
          ELSE
*
*             Reordering successful
*
          IF( WORK( 2, 1 ).EQ.ZERO ) THEN
*
*             Form C = T22 - lambda*I in WORK(2:N,2:N).
*
              DO 20 I = 2, N
                  WORK( I, I ) = WORK( I, I ) - WORK( 1, 1 )
20             CONTINUE
              N2 = 1
              NN = N - 1
          ELSE
*
*             Triangularize the 2 by 2 block by unitary
*             transformation U = [  cs  i*ss ]
*                               [ i*ss  cs  ].
*             such that the (1,1) position of WORK is complex
*             eigenvalue lambda with positive imaginary part. (2,2)
*             position of WORK is the complex eigenvalue lambda

```

```

*           with negative imaginary part.
*
*           MU = SQRT( ABS( WORK( 1, 2 ) ) ) *
$           SQRT( ABS( WORK( 2, 1 ) ) )
*           DELTA = DLAPY2( MU, WORK( 2, 1 ) )
*           CS = MU / DELTA
*           SN = -WORK( 2, 1 ) / DELTA
*
*           Form
*
*           C' = WORK(2:N,2:N) + i*[rwork(1) ..... rwork(n-1) ]
*           [      mu      ]
*           [      ..      ]
*           [      ..      ]
*           [      mu      ]
*           where C' is conjugate transpose of complex matrix C,
*           and RWORK is stored starting in the N+1-st column of
*           WORK.
*
*           DO 30 J = 3, N
*               WORK( 2, J ) = CS*WORK( 2, J )
*               WORK( J, J ) = WORK( J, J ) - WORK( 1, 1 )
30          CONTINUE
*           WORK( 2, 2 ) = ZERO
*
*           WORK( 1, N+1 ) = TWO*MU
*           DO 40 I = 2, N - 1
*               WORK( I, N+1 ) = SN*WORK( 1, I+1 )
40          CONTINUE
*           N2 = 2
*           NN = 2*( N-1 )
*           END IF
*
*           Estimate norm(inv(C'))
*
*           EST = ZERO
*           KASE = 0
50          CONTINUE
$          CALL DLACON( NN, WORK( 1, N+2 ), WORK( 1, N+4 ), IWORK,
$                  EST, KASE )
*           IF( KASE.NE.0 ) THEN
*               IF( KASE.EQ.1 ) THEN
*                   IF( N2.EQ.1 ) THEN
*
*                       Real eigenvalue: solve C'*x = scale*c.
*
*                       CALL DLAQTR( .TRUE., .TRUE., N-1, WORK( 2, 2 ),
$                               LDWORK, DUMMY, DUMM, SCALE,
$                               WORK( 1, N+4 ), WORK( 1, N+6 ),
$                               IERR )

```

```

                                ELSE
*
*                                Complex eigenvalue: solve
*                                C*(p+iq) = scale*(c+id) in real arithmetic.
*
                                CALL DLAQTR( .TRUE., .FALSE., N-1, WORK( 2, 2 ),
$                                LDWORK, WORK( 1, N+1 ), MU, SCALE,
$                                WORK( 1, N+4 ), WORK( 1, N+6 ),
$                                IERR )
                                END IF
                                ELSE
                                IF( N2.EQ.1 ) THEN
*
*                                Real eigenvalue: solve C*x = scale*c.
*
                                CALL DLAQTR( .FALSE., .TRUE., N-1, WORK( 2, 2 ),
$                                LDWORK, DUMMY, DUMM, SCALE,
$                                WORK( 1, N+4 ), WORK( 1, N+6 ),
$                                IERR )
                                ELSE
*
*                                Complex eigenvalue: solve
*                                C*(p+iq) = scale*(c+id) in real arithmetic.
*
                                CALL DLAQTR( .FALSE., .FALSE., N-1,
$                                WORK( 2, 2 ), LDWORK,
$                                WORK( 1, N+1 ), MU, SCALE,
$                                WORK( 1, N+4 ), WORK( 1, N+6 ),
$                                IERR )
*
                                END IF
                                END IF
*
                                GO TO 50
                                END IF
                                END IF
*
                                SEP( KS ) = SCALE / MAX( EST, SMLNUM )
                                IF( PAIR )
$                                SEP( KS+1 ) = SEP( KS )
                                END IF
*
                                IF( PAIR )
$                                KS = KS + 1
*
60 CONTINUE
RETURN
*
* End of DTRSNA
*
```

END

— LAPACK dtrsna —

```
(let* ((zero 0.0) (one 1.0) (two 2.0))
  (declare (type (double-float 0.0 0.0) zero)
            (type (double-float 1.0 1.0) one)
            (type (double-float 2.0 2.0) two))
  (defun dtrsna
    (job howmny select n t$ ldt vl ldvl vr ldvr s sep mm m work ldwork
     iwork info)
    (declare (type (simple-array fixnum (*)) iwork)
              (type (simple-array double-float (*)) work sep s vr vl t$)
              (type fixnum info ldwork m mm ldvr ldvl ldt n)
              (type (simple-array (member t nil) (*)) select)
              (type character howmny job))
    (f2cl-lib:with-multi-array-data
      ((job character job-%data% job-%offset%)
       (howmny character howmny-%data% howmny-%offset%)
       (select (member t nil) select-%data% select-%offset%)
       (t$ double-float t$-%data% t$-%offset%)
       (vl double-float vl-%data% vl-%offset%)
       (vr double-float vr-%data% vr-%offset%)
       (s double-float s-%data% s-%offset%)
       (sep double-float sep-%data% sep-%offset%)
       (work double-float work-%data% work-%offset%)
       (iwork fixnum iwork-%data% iwork-%offset%))
      (prog ((dummy (make-array 1 :element-type 'double-float)) (bignum 0.0)
              (cond$ 0.0) (cs 0.0) (delta 0.0) (dumm 0.0) (eps 0.0) (est 0.0)
              (lnrm 0.0) (mu 0.0) (prod 0.0) (prod1 0.0) (prod2 0.0) (rnrm 0.0)
              (scale 0.0) (smlnum 0.0) (sn 0.0) (i 0) (ierr 0) (ifst 0) (ilst 0)
              (j 0) (k 0) (kase 0) (ks 0) (n2 0) (nn 0) (pair nil) (somcon nil)
              (wantbh nil) (wants nil) (wantsp nil) (/=$ 0.0f0))
              (declare (type (single-float) /=$)
                        (type (simple-array double-float (1)) dummy)
                        (type (double-float) bignum cond$ cs delta dumm eps est lnrm
                               mu prod prod1 prod2 rnrm scale smlnum sn)
                        (type fixnum i ierr ifst ilst j k kase ks n2 nn)
                        (type (member t nil) pair somcon wantbh wants wantsp))
              (setf wantbh (char-equal job #\B))
              (setf wants (or (char-equal job #\E) wantbh))
              (setf wantsp (or (char-equal job #\V) wantbh))
              (setf somcon (char-equal howmny #\S))
              (setf info 0)
              (cond
                ((and (not wants) (not wantsp))
```

```

(setf info -1))
((and (not (char-equal howmny #\A)) (not somcon))
 (setf info -2))
((< n 0)
 (setf info -4))
((< ldt (max (the fixnum 1) (the fixnum n)))
 (setf info -6))
((or (< ldvl 1) (and wants (< ldvl n)))
 (setf info -8))
((or (< ldvr 1) (and wants (< ldvr n)))
 (setf info -10))
(t
 (cond
  (somcon
   (setf m 0)
   (setf pair nil)
   (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
    ((> k n) nil)
    (tagbody
     (cond
      (pair
       (setf pair nil))
      (t
       (cond
        ((< k n)
         (cond
          ((=
            (f2cl-lib:fref t$
                          ((f2cl-lib:int-add k 1) k)
                          ((1 ldt) (1 *)))
            zero)
           (if
            (f2cl-lib:fref select-%data%
                          (k)
                          ((1 *))
                          select-%offset%)
            (setf m (f2cl-lib:int-add m 1))))
          (t
           (setf pair t)
           (if
            (or
             (f2cl-lib:fref select-%data%
                           (k)
                           ((1 *))
                           select-%offset%)
             (f2cl-lib:fref select-%data%
                           ((f2cl-lib:int-add k 1)
                           ((1 *))
                           select-%offset%))
            (setf m (f2cl-lib:int-add m 2)))))))

```

```

(t
  (if
    (f2cl-lib:fref select-%data%
                      (n)
                      ((1 *))
                      select-%offset%)
    (setf m (f2cl-lib:int-add m 1))))))
(t
  (setf m n))
(cond
  ((< mm m)
   (setf info -13))
  ((or (< ldwork 1) (and wantsp (< ldwork n)))
   (setf info -16))))
(cond
  ((/= info 0)
   (error
    " ** On entry to ~a parameter number ~a had an illegal value~%"
    "DTRSNA" (f2cl-lib:int-sub info))
   (go end_label)))
(if (= n 0) (go end_label))
(cond
  ((= n 1)
   (cond
    (somcon
     (if
      (not (f2cl-lib:fref select-%data% (1) ((1 *)) select-%offset%))
      (go end_label)))
     (if wants (setf (f2cl-lib:fref s-%data% (1) ((1 *)) s-%offset%) one))
     (if wantsp
      (setf (f2cl-lib:fref sep-%data% (1) ((1 *)) sep-%offset%)
            (abs
             (f2cl-lib:fref t$-%data%
                             (1 1)
                             ((1 ldt) (1 *))
                             t$-%offset%))))
      (go end_label)))
   (setf eps (dlamch "P"))
   (setf smlnum (/ (dlamch "S") eps))
   (setf bignum (/ one smlnum))
   (multiple-value-bind (var-0 var-1)
     (dlabad smlnum bignum)
     (declare (ignore))
     (setf smlnum var-0)
     (setf bignum var-1))
   (setf ks 0)
   (setf pair nil)
   (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
     (> k n) nil)
   (tagbody

```



```

1))
(setf rnorm
  (dnrm2 n
    (f2cl-lib:array-slice vr
      double-float
      (1 ks)
      ((1 ldvr) (1 *)))
    1))
(setf lnrm
  (dnrm2 n
    (f2cl-lib:array-slice vl
      double-float
      (1 ks)
      ((1 ldvl) (1 *)))
    1))
(setf (f2cl-lib:fref s-%data% (ks) ((1 *)) s-%offset%)
  (/ (abs prod) (* rnorm lnrm)))
(t
  (setf prod1
    (ddot n
      (f2cl-lib:array-slice vr
        double-float
        (1 ks)
        ((1 ldvr) (1 *)))
      1
      (f2cl-lib:array-slice vl
        double-float
        (1 ks)
        ((1 ldvl) (1 *)))
      1))
  (setf prod1
    (+ prod1
      (ddot n
        (f2cl-lib:array-slice vr
          double-float
          (1 (f2cl-lib:int-add ks 1))
          ((1 ldvr) (1 *)))
        1
        (f2cl-lib:array-slice vl
          double-float
          (1 (f2cl-lib:int-add ks 1))
          ((1 ldvl) (1 *)))
        1)))
  (setf prod2
    (ddot n
      (f2cl-lib:array-slice vl
        double-float
        (1 ks)
        ((1 ldvl) (1 *)))
      1

```



```

        (f2cl-lib:array-slice vr
                                double-float
                                (1 (f2cl-lib:int-add ks 1))
                                ((1 ldvr) (1 *)))
1))
(setf prod2
  (- prod2
    (ddot n
      (f2cl-lib:array-slice vl
                              double-float
                              (1 (f2cl-lib:int-add ks 1))
                              ((1 ldvl) (1 *)))
      1
      (f2cl-lib:array-slice vr
                              double-float
                              (1 ks)
                              ((1 ldvr) (1 *)))
      1)))
(setf rnorm
  (dlapy2
    (dnrm2 n
      (f2cl-lib:array-slice vr
                              double-float
                              (1 ks)
                              ((1 ldvr) (1 *)))
      1)
    (dnrm2 n
      (f2cl-lib:array-slice vr
                              double-float
                              (1 (f2cl-lib:int-add ks 1))
                              ((1 ldvr) (1 *)))
      1)))
(setf lnrm
  (dlapy2
    (dnrm2 n
      (f2cl-lib:array-slice vl
                              double-float
                              (1 ks)
                              ((1 ldvl) (1 *)))
      1)
    (dnrm2 n
      (f2cl-lib:array-slice vl
                              double-float
                              (1 (f2cl-lib:int-add ks 1))
                              ((1 ldvl) (1 *)))
      1)))
(setf cond$ (/ (dlapy2 prod1 prod2) (* rnorm lnrm)))
(setf (f2cl-lib:fref s-%data% (ks) ((1 *)) s-%offset%) cond$)
(setf (f2cl-lib:fref s-%data%
                    ((f2cl-lib:int-add ks 1))

```

```

                                ((1 *))
                                s-%offset%)
                                cond$))))))
(cond
  (wantsp
    (dlacpy "Full" n n t$ ldt work ldwork)
    (setf ifst k)
    (setf ilst 1)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
       var-9)
      (dtrexc "No Q" n work ldwork dummy 1 ifst ilst
        (f2cl-lib:array-slice work
          double-float
          (1 (f2cl-lib:int-add n 1))
          ((1 ldwork) (1 *))))
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-8))
      (setf ifst var-6)
      (setf ilst var-7)
      (setf ierr var-9))
    (cond
      ((or (= ierr 1) (= ierr 2))
        (setf scale one)
        (setf est bignum))
      (t
        (tagbody
          (cond
            ((= (f2cl-lib:fref work (2 1) ((1 ldwork) (1 *))) zero)
              (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                ((> i n) nil)
              (tagbody
                (setf (f2cl-lib:fref work-%data%
                  (i i)
                  ((1 ldwork) (1 *)))
                  work-%offset%)
                  (-
                    (f2cl-lib:fref work-%data%
                      (i i)
                      ((1 ldwork) (1 *)))
                    work-%offset%)
                    (f2cl-lib:fref work-%data%
                      (1 1)
                      ((1 ldwork) (1 *)))
                    work-%offset%))))))
            (setf n2 1)
            (setf nn (f2cl-lib:int-sub n 1)))
          (t
            (setf mu
              (*

```

```

(f2cl-lib:fsqrt
  (abs
    (f2cl-lib:fref work-%data%
                    (1 2)
                    ((1 ldwork) (1 *))
                    work-%offset%)))
(f2cl-lib:fsqrt
  (abs
    (f2cl-lib:fref work-%data%
                    (2 1)
                    ((1 ldwork) (1 *))
                    work-%offset%))))
(setf delta
  (dlapy2 mu
    (f2cl-lib:fref work-%data%
                    (2 1)
                    ((1 ldwork) (1 *))
                    work-%offset%)))
(setf cs (/ mu delta))
(setf sn
  (/
    (-
      (f2cl-lib:fref work-%data%
                      (2 1)
                      ((1 ldwork) (1 *))
                      work-%offset%))
      delta))
(f2cl-lib:fdo (j 3 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref work-%data%
                      (2 j)
                      ((1 ldwork) (1 *))
                      work-%offset%)
    (* cs
      (f2cl-lib:fref work-%data%
                      (2 j)
                      ((1 ldwork) (1 *))
                      work-%offset%)))
  (setf (f2cl-lib:fref work-%data%
                      (j j)
                      ((1 ldwork) (1 *))
                      work-%offset%)
    (-
      (f2cl-lib:fref work-%data%
                      (j j)
                      ((1 ldwork) (1 *))
                      work-%offset%)
      (f2cl-lib:fref work-%data%
                      (1 1)

```

```

((1 ldwork) (1 *))
work-%offset%))))
(setf (f2cl-lib:fref work-%data%
                    (2 2)
                    ((1 ldwork) (1 *))
                    work-%offset%)
      zero)
(setf (f2cl-lib:fref work-%data%
                    (1 (f2cl-lib:int-add n 1))
                    ((1 ldwork) (1 *))
                    work-%offset%)
      (* two mu))
(f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
              ((> i
                 (f2cl-lib:int-add n
                                     (f2cl-lib:int-sub
                                      1)))
               nil)
  (tagbody
    (setf (f2cl-lib:fref work-%data%
                        (i (f2cl-lib:int-add n 1))
                        ((1 ldwork) (1 *))
                        work-%offset%)
          (* sn
             (f2cl-lib:fref work-%data%
                             (1 (f2cl-lib:int-add i 1))
                             ((1 ldwork) (1 *))
                             work-%offset%))))))
(setf n2 2)
(setf nn (f2cl-lib:int-mul 2 (f2cl-lib:int-sub n 1))))
(setf est zero)
(setf kase 0)

label50
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
  (dlacon nn
    (f2cl-lib:array-slice work
                          double-float
                          (1 (f2cl-lib:int-add n 2))
                          ((1 ldwork) (1 *)))
    (f2cl-lib:array-slice work
                          double-float
                          (1 (f2cl-lib:int-add n 4))
                          ((1 ldwork) (1 *)))

    iwork est kase)
  (declare (ignore var-0 var-1 var-2 var-3))
  (setf est var-4)
  (setf kase var-5))
(cond
  ((/= kase 0)
   (cond

```

```

(= kase 1)
(cond
  (= n2 1)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10)
      (dlaqtr t t
        (f2cl-lib:int-sub n 1)
        (f2cl-lib:array-slice work
          double-float
          (2 2)
          ((1 ldwork) (1 *)))
        ldwork dummy dumm scale
        (f2cl-lib:array-slice work
          double-float
          (1
            (f2cl-lib:int-add n
              4))
            ((1 ldwork) (1 *)))
        (f2cl-lib:array-slice work
          double-float
          (1
            (f2cl-lib:int-add n
              6))
            ((1 ldwork) (1 *)))
        ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4
        var-5 var-6 var-8 var-9))
      (setf scale var-7)
      (setf ierr var-10)))
  (t
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10)
      (dlaqtr t nil
        (f2cl-lib:int-sub n 1)
        (f2cl-lib:array-slice work
          double-float
          (2 2)
          ((1 ldwork) (1 *)))
        ldwork
        (f2cl-lib:array-slice work
          double-float
          (1
            (f2cl-lib:int-add n
              1))
            ((1 ldwork) (1 *)))
        mu scale
        (f2cl-lib:array-slice work
          double-float

```

```

                                (1
                                (f2cl-lib:int-add n
                                                4))
                                ((1 ldwork) (1 *)))
(f2cl-lib:array-slice work
double-float
(1
(f2cl-lib:int-add n
                                                6))
((1 ldwork) (1 *)))

ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4
var-5 var-6 var-8 var-9))
(setf scale var-7)
(setf ierr var-10))))))
(t
(cond
(= n2 1)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6
var-7 var-8 var-9 var-10)
(dlaqtr nil t
(f2cl-lib:int-sub n 1)
(f2cl-lib:array-slice work
double-float
(2 2)
((1 ldwork) (1 *)))
ldwork dummy dumm scale
(f2cl-lib:array-slice work
double-float
(1
(f2cl-lib:int-add n
                                                4))
((1 ldwork) (1 *)))
(f2cl-lib:array-slice work
double-float
(1
(f2cl-lib:int-add n
                                                6))
((1 ldwork) (1 *)))

ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4
var-5 var-6 var-8 var-9))
(setf scale var-7)
(setf ierr var-10))))
(t
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6
var-7 var-8 var-9 var-10)
(dlaqtr nil nil

```

```
(f2cl-lib:int-sub n 1)
(f2cl-lib:array-slice work
                        double-float
                        (2 2)
                        ((1 ldwork) (1 *)))

ldwork
(f2cl-lib:array-slice work
                        double-float
                        (1
                         (f2cl-lib:int-add n
                                              1))
                        ((1 ldwork) (1 *)))

mu scale
(f2cl-lib:array-slice work
                        double-float
                        (1
                         (f2cl-lib:int-add n
                                              4))
                        ((1 ldwork) (1 *)))

(f2cl-lib:array-slice work
                        double-float
                        (1
                         (f2cl-lib:int-add n
                                              6))
                        ((1 ldwork) (1 *)))

ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4
                 var-5 var-6 var-8 var-9))
(setf scale var-7)
(setf ierr var-10))))))
(go label50))))))
(setf (f2cl-lib:fref sep-%data% (ks) ((1 *)) sep-%offset%)
      (/ scale (max est smlnum)))
(if pair
    (setf (f2cl-lib:fref sep-%data%
                          ((f2cl-lib:int-add ks 1))
                          ((1 *))
                          sep-%offset%)
          (f2cl-lib:fref sep-%data%
                          (ks)
                          ((1 *))
                          sep-%offset%))))
    (if pair (setf ks (f2cl-lib:int-add ks 1)))
label60))
end_label

(return
 (values nil
         nil
         nil
         nil)
```

```

nil
nil
nil
nil
nil
nil
nil
nil
nil
nil
m
nil
nil
nil
info))))))

```

—————→

ieeeck LAPACK

— ieeeck.input —

```

)set break resume
)sys rm -f ieeeck.output
)spool ieeeck.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

—————→

— ieeeck.help —

```

=====
ieeeck examples
=====

=====
Man Page Details
=====

```

NAME

IEEECK - called from the ILAENV to verify that Infinity and possibly

NaN arithmetic is safe (i.e

SYNOPSIS

```
INTEGER FUNCTION IEEECK( ISPEC, ZERO, ONE )
```

```
    INTEGER    ISPEC
```

```
    REAL       ONE, ZERO
```

Purpose

```
=====
```

IEEECK is called from the ILAENV to verify that Infinity and possibly NaN arithmetic is safe (i.e. will not trap).

Arguments

```
=====
```

ISPEC (input) INTEGER
Specifies whether to test just for infinity arithmetic or whether to test for infinity and NaN arithmetic.
= 0: Verify infinity arithmetic only.
= 1: Verify infinity and NaN arithmetic.

ZERO (input) REAL
Must contain the value 0.0
This is passed to prevent the compiler from optimizing away this code.

ONE (input) REAL
Must contain the value 1.0
This is passed to prevent the compiler from optimizing away this code.

RETURN VALUE: INTEGER
= 0: Arithmetic failed to produce the correct answers
= 1: Arithmetic produced the correct answers

— ieeeck.f —

```
INTEGER    FUNCTION IEEECK( ISPEC, ZERO, ONE )
```

```
*
* -- LAPACK auxiliary routine (version 3.0) --
*   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*   Courant Institute, Argonne National Lab, and Rice University
*   June 30, 1998
```

```

*
*   .. Scalar Arguments ..
      INTEGER          ISPEC
      REAL             ONE, ZERO
*
*   ..
*
*   .. Local Scalars ..
      REAL             NAN1, NAN2, NAN3, NAN4, NAN5, NAN6, NEGINF,
$                     NEGZRO, NEWZRO, POSINF
*
*   .. Executable Statements ..
      IEEECK = 1
*
      POSINF = ONE / ZERO
      IF( POSINF.LE.ONE ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      NEGINF = -ONE / ZERO
      IF( NEGINF.GE.ZERO ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      NEGZRO = ONE / ( NEGINF+ONE )
      IF( NEGZRO.NE.ZERO ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      NEGINF = ONE / NEGZRO
      IF( NEGINF.GE.ZERO ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      NEWZRO = NEGZRO + ZERO
      IF( NEWZRO.NE.ZERO ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      POSINF = ONE / NEWZRO
      IF( POSINF.LE.ONE ) THEN
         IEEECK = 0
         RETURN
      END IF
*
      NEGINF = NEGINF*POSINF

```

```

      IF( NEGINF.GE.ZERO ) THEN
        IEEECK = 0
        RETURN
      END IF
*
      POSINF = POSINF*POSINF
      IF( POSINF.LE.ONE ) THEN
        IEEECK = 0
        RETURN
      END IF
*
*
*
*
*      Return if we were only asked to check infinity arithmetic
*
      IF( ISPEC.EQ.0 )
$      RETURN
*
      NAN1 = POSINF + NEGINF
*
      NAN2 = POSINF / NEGINF
*
      NAN3 = POSINF / POSINF
*
      NAN4 = POSINF*ZERO
*
      NAN5 = NEGINF*NEGZRO
*
      NAN6 = NAN5*0.0
*
      IF( NAN1.EQ.NAN1 ) THEN
        IEEECK = 0
        RETURN
      END IF
*
      IF( NAN2.EQ.NAN2 ) THEN
        IEEECK = 0
        RETURN
      END IF
*
      IF( NAN3.EQ.NAN3 ) THEN
        IEEECK = 0
        RETURN
      END IF
*
      IF( NAN4.EQ.NAN4 ) THEN
        IEEECK = 0
        RETURN
      END IF

```

```

*
  IF( NAN5.EQ.NAN5 ) THEN
    IEEECK = 0
    RETURN
  END IF
*
  IF( NAN6.EQ.NAN6 ) THEN
    IEEECK = 0
    RETURN
  END IF
*
  RETURN
END

```

— LAPACK ieeeck —

```

(defun ieeeck (ispec zero one)
  (declare (type (single-float) one zero) (type fixnum ispec))
  (prog ((nan1 0.0f0) (nan2 0.0f0) (nan3 0.0f0) (nan4 0.0f0) (nan5 0.0f0)
        (nan6 0.0f0) (neginf 0.0f0) (negzro 0.0f0) (newzro 0.0f0)
        (posinf 0.0f0) (ieeeck 0))
    (declare (type fixnum ieeeck)
      (type (single-float) posinf newzro negzro neginf nan6 nan5 nan4
        nan3 nan2 nan1))

    (setf ieeeck 1)
    (setf posinf (/ one zero))
    (cond
      ((<= posinf one)
        (setf ieeeck 0)
        (go end_label)))
    (setf neginf (/ (- one) zero))
    (cond
      ((>= neginf zero)
        (setf ieeeck 0)
        (go end_label)))
    (setf negzro (/ one (+ neginf one)))
    (cond
      ((/= negzro zero)
        (setf ieeeck 0)
        (go end_label)))
    (setf neginf (/ one negzro))
    (cond
      ((>= neginf zero)
        (setf ieeeck 0)
        (go end_label)))
    (setf newzro (+ negzro zero))

```

```

(cond
  ((/= newzro zero)
   (setf ieeeck 0)
   (go end_label)))
(setf posinf (/ one newzro))
(cond
  ((<= posinf one)
   (setf ieeeck 0)
   (go end_label)))
(setf neginf (* neginf posinf))
(cond
  ((>= neginf zero)
   (setf ieeeck 0)
   (go end_label)))
(setf posinf (* posinf posinf))
(cond
  ((<= posinf one)
   (setf ieeeck 0)
   (go end_label)))
(if (= ispec 0) (go end_label))
(setf nan1 (+ posinf neginf))
(setf nan2 (/ posinf neginf))
(setf nan3 (/ posinf posinf))
(setf nan4 (* posinf zero))
(setf nan5 (* neginf negzro))
(setf nan6 (* nan5 0.0f0))
(cond
  ((= nan1 nan1)
   (setf ieeeck 0)
   (go end_label)))
(cond
  ((= nan2 nan2)
   (setf ieeeck 0)
   (go end_label)))
(cond
  ((= nan3 nan3)
   (setf ieeeck 0)
   (go end_label)))
(cond
  ((= nan4 nan4)
   (setf ieeeck 0)
   (go end_label)))
(cond
  ((= nan5 nan5)
   (setf ieeeck 0)
   (go end_label)))
(cond
  ((= nan6 nan6)
   (setf ieeeck 0)
   (go end_label)))

```

```

end_label
  (return (values iieeeck nil nil nil)))

```

ilaenv LAPACK

— ilaenv.input —

```

)set break resume
)sys rm -f ilaenv.output
)spool ilaenv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ilaenv.help —

```

=====
ilaenv examples
=====

```

```

=====
Man Page Details
=====

```

NAME

ILAENV - called from the LAPACK routines to choose problem-dependent parameters for the local environment

SYNOPSIS

```

INTEGER FUNCTION ILAENV( ISPEC, NAME, OPTS, N1, N2, N3, N4 )

```

```

CHARACTER*( * ) NAME, OPTS

```

```

INTEGER      ISPEC, N1, N2, N3, N4

```

Purpose

```

=====

```

ILAENV is called from the LAPACK routines to choose problem-dependent parameters for the local environment. See ISPEC for a description of the parameters.

This version provides a set of parameters which should give good, but not optimal, performance on many of the currently available computers. Users are encouraged to modify this subroutine to set the tuning parameters for their particular machine using the option and problem size information in the arguments.

This routine will not function correctly if it is converted to all lower case. Converting it to all upper case is allowed.

Arguments

=====

ISPEC (input) INTEGER
 Specifies the parameter to be returned as the value of ILAENV.

- = 1: the optimal blocksize; if this value is 1, an unblocked algorithm will give the best performance.
- = 2: the minimum block size for which the block routine should be used; if the usable block size is less than this value, an unblocked routine should be used.
- = 3: the crossover point (in a block routine, for N less than this value, an unblocked routine should be used)
- = 4: the number of shifts, used in the nonsymmetric eigenvalue routines
- = 5: the minimum column dimension for blocking to be used; rectangular blocks must have dimension at least k by m, where k is given by ILAENV(2,...) and m by ILAENV(5,...)
- = 6: the crossover point for the SVD (when reducing an m by n matrix to bidiagonal form, if $\max(m,n)/\min(m,n)$ exceeds this value, a QR factorization is used first to reduce the matrix to a triangular form.)
- = 7: the number of processors
- = 8: the crossover point for the multishift QR and QZ methods for nonsymmetric eigenvalue problems.
- = 9: maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm (used by xGELSD and xGESDD)
- =10: ieee NaN arithmetic can be trusted not to trap
- =11: infinity arithmetic can be trusted not to trap

NAME (input) CHARACTER*(*)
 The name of the calling subroutine, in either upper case or lower case.

OPTS (input) CHARACTER*(*)
 The character options to the subroutine NAME, concatenated

into a single character string. For example, UPLO = 'U', TRANS = 'T', and DIAG = 'N' for a triangular routine would be specified as OPTS = 'UTN'.

N1 (input) INTEGER
 N2 (input) INTEGER
 N3 (input) INTEGER
 N4 (input) INTEGER
 Problem dimensions for the subroutine NAME; these may not all be required.

(ILAENV) (output) INTEGER
 >= 0: the value of the parameter specified by ISPEC
 < 0: if ILAENV = -k, the k-th argument had an illegal value.

Further Details =====

The following conventions have been used when calling ILAENV from the LAPACK routines:

- 1) OPTS is a concatenation of all of the character options to subroutine NAME, in the same order that they appear in the argument list for NAME, even if they are not used in determining the value of the parameter specified by ISPEC.
- 2) The problem dimensions N1, N2, N3, N4 are specified in the order that they appear in the argument list for NAME. N1 is used first, N2 second, and so on, and unused problem dimensions are passed a value of -1.
- 3) The parameter value returned by ILAENV is checked for validity in the calling subroutine. For example, ILAENV is used to retrieve the optimal blocksize for STRTRI as follows:

```
NB = ILAENV( 1, 'STRTRI', UPLO // DIAG, N, -1, -1, -1 )
IF( NB.LE.1 ) NB = MAX( 1, N )
```

— ilaenv.f —

```
      INTEGER      FUNCTION ILAENV( ISPEC, NAME, OPTS, N1, N2, N3,
$      N4 )

*
*  -- LAPACK auxiliary routine (version 3.0) --
*  Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*  Courant Institute, Argonne National Lab, and Rice University
*  June 30, 1999
*
*  .. Scalar Arguments ..
```



```

      CHARACTER*( * )   NAME, OPTS
      INTEGER           ISPEC, N1, N2, N3, N4
*
*   ..
*
*   =====
*
*   .. Local Scalars ..
      LOGICAL           CNAME, SNAME
      CHARACTER*1        C1
      CHARACTER*2        C2, C4
      CHARACTER*3        C3
      CHARACTER*6        SUBNAM
      INTEGER           I, IC, IZ, NB, NBMIN, NX
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC          CHAR, ICHAR, INT, MIN, REAL
*
*   ..
*   .. External Functions ..
      INTEGER           IEEECK
      EXTERNAL           IEEECK
*
*   ..
*   .. Executable Statements ..
*
      GO TO ( 100, 100, 100, 400, 500, 600, 700, 800, 900, 1000,
$           1100 ) ISPEC
*
*   Invalid value for ISPEC
*
*
      ILAENV = -1
      RETURN
*
100 CONTINUE
*
*   Convert NAME to upper case if the first character is lower case.
*
      ILAENV = 1
      SUBNAM = NAME
      IC = ICHAR( SUBNAM( 1:1 ) )
      IZ = ICHAR( 'Z' )
      IF( IZ.EQ.90 .OR. IZ.EQ.122 ) THEN
*
*   ASCII character set
*
*
      IF( IC.GE.97 .AND. IC.LE.122 ) THEN
        SUBNAM( 1:1 ) = CHAR( IC-32 )
        DO 10 I = 2, 6
          IC = ICHAR( SUBNAM( I:I ) )
          IF( IC.GE.97 .AND. IC.LE.122 )
$            SUBNAM( I:I ) = CHAR( IC-32 )
10      CONTINUE

```

```

        END IF
*
        ELSE IF( IZ.EQ.233 .OR. IZ.EQ.169 ) THEN
*
*        EBCDIC character set
*
        IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR.
$      ( IC.GE.145 .AND. IC.LE.153 ) .OR.
$      ( IC.GE.162 .AND. IC.LE.169 ) ) THEN
            SUBNAM( 1:1 ) = CHAR( IC+64 )
            DO 20 I = 2, 6
                IC = ICHAR( SUBNAM( I:I ) )
                IF( ( IC.GE.129 .AND. IC.LE.137 ) .OR.
$              ( IC.GE.145 .AND. IC.LE.153 ) .OR.
$              ( IC.GE.162 .AND. IC.LE.169 ) )
$              SUBNAM( I:I ) = CHAR( IC+64 )
20          CONTINUE
            END IF
*
        ELSE IF( IZ.EQ.218 .OR. IZ.EQ.250 ) THEN
*
*        Prime machines: ASCII+128
*
        IF( IC.GE.225 .AND. IC.LE.250 ) THEN
            SUBNAM( 1:1 ) = CHAR( IC-32 )
            DO 30 I = 2, 6
                IC = ICHAR( SUBNAM( I:I ) )
                IF( IC.GE.225 .AND. IC.LE.250 )
$              SUBNAM( I:I ) = CHAR( IC-32 )
30          CONTINUE
            END IF
        END IF
*
        C1 = SUBNAM( 1:1 )
        SNAME = C1.EQ.'S' .OR. C1.EQ.'D'
        CNAME = C1.EQ.'C' .OR. C1.EQ.'Z'
        IF( .NOT.( CNAME .OR. SNAME ) )
$      RETURN
        C2 = SUBNAM( 2:3 )
        C3 = SUBNAM( 4:6 )
        C4 = C3( 2:3 )
*
        GO TO ( 110, 200, 300 ) ISPEC
*
110 CONTINUE
*
        ISPEC = 1: block size
*
*
*      In these examples, separate code is provided for setting NB for
*      real and complex. We assume that NB will take the same value in

```

```

*      single or double precision.
*
      NB = 1
*
      IF( C2.EQ.'GE' ) THEN
        IF( C3.EQ.'TRF' ) THEN
          IF( SNAME ) THEN
            NB = 64
          ELSE
            NB = 64
          END IF
        ELSE IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR.
$          C3.EQ.'QLF' ) THEN
          IF( SNAME ) THEN
            NB = 32
          ELSE
            NB = 32
          END IF
        ELSE IF( C3.EQ.'HRD' ) THEN
          IF( SNAME ) THEN
            NB = 32
          ELSE
            NB = 32
          END IF
        ELSE IF( C3.EQ.'BRD' ) THEN
          IF( SNAME ) THEN
            NB = 32
          ELSE
            NB = 32
          END IF
        ELSE IF( C3.EQ.'TRI' ) THEN
          IF( SNAME ) THEN
            NB = 64
          ELSE
            NB = 64
          END IF
        END IF
      ELSE IF( C2.EQ.'PO' ) THEN
        IF( C3.EQ.'TRF' ) THEN
          IF( SNAME ) THEN
            NB = 64
          ELSE
            NB = 64
          END IF
        END IF
      ELSE IF( C2.EQ.'SY' ) THEN
        IF( C3.EQ.'TRF' ) THEN
          IF( SNAME ) THEN
            NB = 64
          ELSE

```

```

        NB = 64
    END IF
    ELSE IF( SNAME .AND. C3.EQ.'TRD' ) THEN
        NB = 32
    ELSE IF( SNAME .AND. C3.EQ.'GST' ) THEN
        NB = 64
    END IF
    ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
        IF( C3.EQ.'TRF' ) THEN
            NB = 64
        ELSE IF( C3.EQ.'TRD' ) THEN
            NB = 32
        ELSE IF( C3.EQ.'GST' ) THEN
            NB = 64
        END IF
    ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$           C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$           C4.EQ.'BR' ) THEN
                NB = 32
            END IF
        ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$           C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$           C4.EQ.'BR' ) THEN
                NB = 32
            END IF
        END IF
    ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$           C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$           C4.EQ.'BR' ) THEN
                NB = 32
            END IF
        ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$           C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$           C4.EQ.'BR' ) THEN
                NB = 32
            END IF
        END IF
    ELSE IF( C2.EQ.'GB' ) THEN
        IF( C3.EQ.'TRF' ) THEN
            IF( SNAME ) THEN
                IF( N4.LE.64 ) THEN
                    NB = 1
                ELSE
                    NB = 32
                END IF
            END IF
        END IF
    END IF

```

```
        END IF
      ELSE
        IF( N4.LE.64 ) THEN
          NB = 1
        ELSE
          NB = 32
        END IF
      END IF
    END IF
  ELSE IF( C2.EQ.'PB' ) THEN
    IF( C3.EQ.'TRF' ) THEN
      IF( SNAME ) THEN
        IF( N2.LE.64 ) THEN
          NB = 1
        ELSE
          NB = 32
        END IF
      ELSE
        IF( N2.LE.64 ) THEN
          NB = 1
        ELSE
          NB = 32
        END IF
      END IF
    ELSE IF( C2.EQ.'TR' ) THEN
      IF( C3.EQ.'TRI' ) THEN
        IF( SNAME ) THEN
          NB = 64
        ELSE
          NB = 64
        END IF
      END IF
    ELSE IF( C2.EQ.'LA' ) THEN
      IF( C3.EQ.'UUM' ) THEN
        IF( SNAME ) THEN
          NB = 64
        ELSE
          NB = 64
        END IF
      END IF
    ELSE IF( SNAME .AND. C2.EQ.'ST' ) THEN
      IF( C3.EQ.'EBZ' ) THEN
        NB = 1
      END IF
    END IF
  END IF
ILAENV = NB
RETURN
```

*

200 CONTINUE

```

*
*   ISPEC = 2:  minimum block size
*
      NBMIN = 2
      IF( C2.EQ.'GE' ) THEN
        IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR.
$       C3.EQ.'QLF' ) THEN
          IF( SNAME ) THEN
            NBMIN = 2
          ELSE
            NBMIN = 2
          END IF
        ELSE IF( C3.EQ.'HRD' ) THEN
          IF( SNAME ) THEN
            NBMIN = 2
          ELSE
            NBMIN = 2
          END IF
        ELSE IF( C3.EQ.'BRD' ) THEN
          IF( SNAME ) THEN
            NBMIN = 2
          ELSE
            NBMIN = 2
          END IF
        ELSE IF( C3.EQ.'TRI' ) THEN
          IF( SNAME ) THEN
            NBMIN = 2
          ELSE
            NBMIN = 2
          END IF
        ELSE IF( C2.EQ.'SY' ) THEN
          IF( C3.EQ.'TRF' ) THEN
            IF( SNAME ) THEN
              NBMIN = 8
            ELSE
              NBMIN = 8
            END IF
          ELSE IF( SNAME .AND. C3.EQ.'TRD' ) THEN
            NBMIN = 2
          END IF
        ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
          IF( C3.EQ.'TRD' ) THEN
            NBMIN = 2
          END IF
        ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
          IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$       C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$       C4.EQ.'BR' ) THEN

```

```

        NBMIN = 2
    END IF
    ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
        IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$         C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$         C4.EQ.'BR' ) THEN
            NBMIN = 2
        END IF
    END IF
    ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
        IF( C3( 1:1 ).EQ.'G' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$         C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$         C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        ELSE IF( C3( 1:1 ).EQ.'M' ) THEN
            IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$         C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$         C4.EQ.'BR' ) THEN
                NBMIN = 2
            END IF
        END IF
    END IF
    ILAENV = NBMIN
    RETURN
*
300 CONTINUE
*
*   ISPEC = 3:  crossover point
*
    NX = 0
    IF( C2.EQ.'GE' ) THEN
        IF( C3.EQ.'QRF' .OR. C3.EQ.'RQF' .OR. C3.EQ.'LQF' .OR.
$     C3.EQ.'QLF' ) THEN
            IF( SNAME ) THEN
                NX = 128
            ELSE
                NX = 128
            END IF
        ELSE IF( C3.EQ.'HRD' ) THEN
            IF( SNAME ) THEN
                NX = 128
            ELSE
                NX = 128
            END IF
        ELSE IF( C3.EQ.'BRD' ) THEN
            IF( SNAME ) THEN
                NX = 128
            ELSE

```

```

        NX = 128
        END IF
    END IF
ELSE IF( C2.EQ.'SY' ) THEN
    IF( SNAME .AND. C3.EQ.'TRD' ) THEN
        NX = 32
    END IF
ELSE IF( CNAME .AND. C2.EQ.'HE' ) THEN
    IF( C3.EQ.'TRD' ) THEN
        NX = 32
    END IF
ELSE IF( SNAME .AND. C2.EQ.'OR' ) THEN
    IF( C3( 1:1 ).EQ.'G' ) THEN
        IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$         C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$         C4.EQ.'BR' ) THEN
            NX = 128
        END IF
    END IF
ELSE IF( CNAME .AND. C2.EQ.'UN' ) THEN
    IF( C3( 1:1 ).EQ.'G' ) THEN
        IF( C4.EQ.'QR' .OR. C4.EQ.'RQ' .OR. C4.EQ.'LQ' .OR.
$         C4.EQ.'QL' .OR. C4.EQ.'HR' .OR. C4.EQ.'TR' .OR.
$         C4.EQ.'BR' ) THEN
            NX = 128
        END IF
    END IF
END IF
END IF
ILAENV = NX
RETURN

*
400 CONTINUE
*
*   ISPEC = 4:  number of shifts (used by xHSEQR)
*
    ILAENV = 6
    RETURN
*
500 CONTINUE
*
*   ISPEC = 5:  minimum column dimension (not used)
*
    ILAENV = 2
    RETURN
*
600 CONTINUE
*
*   ISPEC = 6:  crossover point for SVD (used by xGELSS and xGESVD)
*
    ILAENV = INT( REAL( MIN( N1, N2 ) ) * 1.6E0 )

```



```

      RETURN
*
* 700 CONTINUE
*
*   ISPEC = 7:  number of processors (not used)
*
*   ILAENV = 1
*   RETURN
*
* 800 CONTINUE
*
*   ISPEC = 8:  crossover point for multishift (used by xHSEQR)
*
*   ILAENV = 50
*   RETURN
*
* 900 CONTINUE
*
*   ISPEC = 9:  maximum size of the subproblems at the bottom of the
*               computation tree in the divide-and-conquer algorithm
*               (used by xGELSD and xGESDD)
*
*   ILAENV = 25
*   RETURN
*
* 1000 CONTINUE
*
*   ISPEC = 10: ieee NaN arithmetic can be trusted not to trap
*
*   C   ILAENV = 0
*       ILAENV = 1
*       IF( ILAENV.EQ.1 ) THEN
*         ILAENV = IEEECK( 0, 0.0, 1.0 )
*       END IF
*       RETURN
*
* 1100 CONTINUE
*
*   ISPEC = 11: infinity arithmetic can be trusted not to trap
*
*   C   ILAENV = 0
*       ILAENV = 1
*       IF( ILAENV.EQ.1 ) THEN
*         ILAENV = IEEECK( 1, 0.0, 1.0 )
*       END IF
*       RETURN
*
*   End of ILAENV
*
END

```

— LAPACK ilaenv —

```

(defun ilaenv (ispec name opts n1 n2 n3 n4)
  (declare (type character opts name)
            (type fixnum n4 n3 n2 n1 ispec))
  (f2cl-lib:with-multi-array-data
    ((name character name-%data% name-%offset%)
     (opts character opts-%data% opts-%offset%))
    (prog ((i 0) (ic 0) (iz 0) (nb 0) (nbmin 0) (nx 0)
           (subnam
            (make-array '(6) :element-type 'character :initial-element #\ ))
            (c3 (make-array '(3) :element-type 'character :initial-element #\ ))
            (c2 (make-array '(2) :element-type 'character :initial-element #\ ))
            (c4 (make-array '(2) :element-type 'character :initial-element #\ ))
            (c1 (make-array '(1) :element-type 'character :initial-element #\ ))
            (cname nil) (sname nil) (ilaenv 0) (char$ 0.0f0))
           (declare (type (single-float) char$)
                     (type (member t nil) sname cname)
                     (type (simple-array character (1)) c1)
                     (type (simple-array character (2)) c4 c2)
                     (type (simple-array character (3)) c3)
                     (type (simple-array character (6)) subnam)
                     (type fixnum ilaenv nx nbmin nb iz ic i))
           (f2cl-lib:computed-goto
            (label100 label100 label100 label1400 label1500 label1600 label1700 label1800
             label1900 label11000 label11100)
            ispec)
           (setf ilaenv -1)
           (go end_label)
           label100
            (setf ilaenv 1)
            (f2cl-lib:f2cl-set-string subnam name (string 6))
            (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (1 1))))
            (setf iz (f2cl-lib:ichar "Z"))
            (cond
              ((or (= iz 90) (= iz 122))
               (cond
                 ((and (>= ic 97) (<= ic 122))
                  (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                                         (code-char (f2cl-lib:int-sub ic 32))))
                 (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                               ((> i 6) nil)
                               (tagbody
                                (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
                                (if (and (>= ic 97) (<= ic 122))

```

```

                (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                                      (code-char
                                       (f2cl-lib:int-sub ic 32)))))))))
((or (= iz 233) (= iz 169))
 (cond
  ((or (and (>= ic 129) (<= ic 137))
        (and (>= ic 145) (<= ic 153))
        (and (>= ic 162) (<= ic 169)))
   (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                         (code-char (f2cl-lib:int-add ic 64)))
   (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                 ((> i 6) nil)
   (tagbody
    (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
    (if
     (or (and (>= ic 129) (<= ic 137))
         (and (>= ic 145) (<= ic 153))
         (and (>= ic 162) (<= ic 169)))
     (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                           (code-char
                            (f2cl-lib:int-add ic 64)))))))))
((or (= iz 218) (= iz 250))
 (cond
  ((and (>= ic 225) (<= ic 250))
   (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (1 1))
                         (code-char (f2cl-lib:int-sub ic 32)))
   (f2cl-lib:fdo (i 2 (f2cl-lib:int-add i 1))
                 ((> i 6) nil)
   (tagbody
    (setf ic (f2cl-lib:ichar (f2cl-lib:fref-string subnam (i i))))
    (if (and (>= ic 225) (<= ic 250))
        (f2cl-lib:fset-string (f2cl-lib:fref-string subnam (i i))
                              (code-char
                               (f2cl-lib:int-sub ic 32)))))))))
(f2cl-lib:f2cl-set-string c1
                          (f2cl-lib:fref-string subnam (1 1))
                          (string 1))
(setf sname (or (f2cl-lib:fstring== c1 "S") (f2cl-lib:fstring== c1 "D")))
(setf cname (or (f2cl-lib:fstring== c1 "C") (f2cl-lib:fstring== c1 "Z")))
(if (not (or cname sname)) (go end_label))
(f2cl-lib:f2cl-set-string c2
                          (f2cl-lib:fref-string subnam (2 3))
                          (string 2))
(f2cl-lib:f2cl-set-string c3
                          (f2cl-lib:fref-string subnam (4 6))
                          (string 3))
(f2cl-lib:f2cl-set-string c4 (f2cl-lib:fref-string c3 (2 3)) (string 2))
(f2cl-lib:computed-goto (label110 label200 label300) ispec)
label110
  (setf nb 1)

```

```

(cond
  ((f2cl-lib:fstring-= c2 "GE")
    (cond
      ((f2cl-lib:fstring-= c3 "TRF")
        (cond
          (sname
            (setf nb 64))
          (t
            (setf nb 64))))))
    ((or (f2cl-lib:fstring-= c3 "QRF")
          (f2cl-lib:fstring-= c3 "RQF")
          (f2cl-lib:fstring-= c3 "LQF")
          (f2cl-lib:fstring-= c3 "QLF"))
      (cond
        (sname
          (setf nb 32))
        (t
          (setf nb 32))))))
    ((f2cl-lib:fstring-= c3 "HRD")
      (cond
        (sname
          (setf nb 32))
        (t
          (setf nb 32))))))
    ((f2cl-lib:fstring-= c3 "BRD")
      (cond
        (sname
          (setf nb 32))
        (t
          (setf nb 32))))))
    ((f2cl-lib:fstring-= c3 "TRI")
      (cond
        (sname
          (setf nb 64))
        (t
          (setf nb 64))))))
    ((f2cl-lib:fstring-= c2 "P0")
      (cond
        ((f2cl-lib:fstring-= c3 "TRF")
          (cond
            (sname
              (setf nb 64))
            (t
              (setf nb 64))))))
          ((f2cl-lib:fstring-= c2 "SY")
            (cond
              ((f2cl-lib:fstring-= c3 "TRF")
                (cond
                  (sname
                    (setf nb 64))

```

```

      (t
        (setf nb 64))))
      ((and sname (f2cl-lib:fstring-= c3 "TRD"))
        (setf nb 32))
      ((and sname (f2cl-lib:fstring-= c3 "GST"))
        (setf nb 64))))
      ((and cname (f2cl-lib:fstring-= c2 "HE"))
        (cond
          ((f2cl-lib:fstring-= c3 "TRF")
            (setf nb 64))
          ((f2cl-lib:fstring-= c3 "TRD")
            (setf nb 32))
          ((f2cl-lib:fstring-= c3 "GST")
            (setf nb 64))))
      ((and sname (f2cl-lib:fstring-= c2 "OR"))
        (cond
          ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
            (cond
              ((or (f2cl-lib:fstring-= c4 "QR")
                    (f2cl-lib:fstring-= c4 "RQ")
                    (f2cl-lib:fstring-= c4 "LQ")
                    (f2cl-lib:fstring-= c4 "QL")
                    (f2cl-lib:fstring-= c4 "HR")
                    (f2cl-lib:fstring-= c4 "TR")
                    (f2cl-lib:fstring-= c4 "BR"))
                (setf nb 32))))
            ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
              (cond
                ((or (f2cl-lib:fstring-= c4 "QR")
                      (f2cl-lib:fstring-= c4 "RQ")
                      (f2cl-lib:fstring-= c4 "LQ")
                      (f2cl-lib:fstring-= c4 "QL")
                      (f2cl-lib:fstring-= c4 "HR")
                      (f2cl-lib:fstring-= c4 "TR")
                      (f2cl-lib:fstring-= c4 "BR"))
                  (setf nb 32))))))
          ((and cname (f2cl-lib:fstring-= c2 "UN"))
            (cond
              ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
                (cond
                  ((or (f2cl-lib:fstring-= c4 "QR")
                        (f2cl-lib:fstring-= c4 "RQ")
                        (f2cl-lib:fstring-= c4 "LQ")
                        (f2cl-lib:fstring-= c4 "QL")
                        (f2cl-lib:fstring-= c4 "HR")
                        (f2cl-lib:fstring-= c4 "TR")
                        (f2cl-lib:fstring-= c4 "BR"))
                      (setf nb 32))))
                ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
                  (cond

```

```

      ((or (f2cl-lib:fstring-= c4 "QR")
           (f2cl-lib:fstring-= c4 "RQ")
           (f2cl-lib:fstring-= c4 "LQ")
           (f2cl-lib:fstring-= c4 "QL")
           (f2cl-lib:fstring-= c4 "HR")
           (f2cl-lib:fstring-= c4 "TR")
           (f2cl-lib:fstring-= c4 "BR")))
      (setf nb 32))))))
((f2cl-lib:fstring-= c2 "GB")
 (cond
  ((f2cl-lib:fstring-= c3 "TRF")
   (cond
    (sname
     (cond
      ((<= n4 64)
       (setf nb 1))
      (t
       (setf nb 32))))
     (t
      (cond
       ((<= n4 64)
        (setf nb 1))
       (t
        (setf nb 32))))))))))
((f2cl-lib:fstring-= c2 "PB")
 (cond
  ((f2cl-lib:fstring-= c3 "TRF")
   (cond
    (sname
     (cond
      ((<= n2 64)
       (setf nb 1))
      (t
       (setf nb 32))))
     (t
      (cond
       ((<= n2 64)
        (setf nb 1))
       (t
        (setf nb 32))))))))))
((f2cl-lib:fstring-= c2 "TR")
 (cond
  ((f2cl-lib:fstring-= c3 "TRI")
   (cond
    (sname
     (setf nb 64))
    (t
     (setf nb 64))))))
((f2cl-lib:fstring-= c2 "LA")
 (cond

```

```

      ((f2cl-lib:fstring== c3 "UUM")
      (cond
        (sname
          (setf nb 64))
        (t
          (setf nb 64))))))
      ((and sname (f2cl-lib:fstring== c2 "ST"))
      (cond
        ((f2cl-lib:fstring== c3 "EBZ")
          (setf nb 1))))))
      (setf ilaenv nb)
      (go end_label)
label200
      (setf nbmin 2)
      (cond
        ((f2cl-lib:fstring== c2 "GE")
          (cond
            ((or (f2cl-lib:fstring== c3 "QRF")
                  (f2cl-lib:fstring== c3 "RQF")
                  (f2cl-lib:fstring== c3 "LQF")
                  (f2cl-lib:fstring== c3 "QLF"))
              (cond
                (sname
                  (setf nbmin 2))
                (t
                  (setf nbmin 2))))
            ((f2cl-lib:fstring== c3 "HRD")
              (cond
                (sname
                  (setf nbmin 2))
                (t
                  (setf nbmin 2))))
            ((f2cl-lib:fstring== c3 "BRD")
              (cond
                (sname
                  (setf nbmin 2))
                (t
                  (setf nbmin 2))))
            ((f2cl-lib:fstring== c3 "TRI")
              (cond
                (sname
                  (setf nbmin 2))
                (t
                  (setf nbmin 2))))))
        ((f2cl-lib:fstring== c2 "SY")
          (cond
            ((f2cl-lib:fstring== c3 "TRF")
              (cond
                (sname
                  (setf nbmin 8))

```

```

(t
  (setf nbmin 8)))
((and sname (f2cl-lib:fstring-= c3 "TRD"))
  (setf nbmin 2)))
((and cname (f2cl-lib:fstring-= c2 "HE"))
  (cond
    ((f2cl-lib:fstring-= c3 "TRD")
      (setf nbmin 2))))
((and sname (f2cl-lib:fstring-= c2 "OR"))
  (cond
    ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
      (cond
        ((or (f2cl-lib:fstring-= c4 "QR")
              (f2cl-lib:fstring-= c4 "RQ")
              (f2cl-lib:fstring-= c4 "LQ")
              (f2cl-lib:fstring-= c4 "QL")
              (f2cl-lib:fstring-= c4 "HR")
              (f2cl-lib:fstring-= c4 "TR")
              (f2cl-lib:fstring-= c4 "BR"))
          (setf nbmin 2))))
      ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
        (cond
          ((or (f2cl-lib:fstring-= c4 "QR")
                (f2cl-lib:fstring-= c4 "RQ")
                (f2cl-lib:fstring-= c4 "LQ")
                (f2cl-lib:fstring-= c4 "QL")
                (f2cl-lib:fstring-= c4 "HR")
                (f2cl-lib:fstring-= c4 "TR")
                (f2cl-lib:fstring-= c4 "BR"))
            (setf nbmin 2)))))))
((and cname (f2cl-lib:fstring-= c2 "UN"))
  (cond
    ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
      (cond
        ((or (f2cl-lib:fstring-= c4 "QR")
              (f2cl-lib:fstring-= c4 "RQ")
              (f2cl-lib:fstring-= c4 "LQ")
              (f2cl-lib:fstring-= c4 "QL")
              (f2cl-lib:fstring-= c4 "HR")
              (f2cl-lib:fstring-= c4 "TR")
              (f2cl-lib:fstring-= c4 "BR"))
          (setf nbmin 2))))
      ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "M")
        (cond
          ((or (f2cl-lib:fstring-= c4 "QR")
                (f2cl-lib:fstring-= c4 "RQ")
                (f2cl-lib:fstring-= c4 "LQ")
                (f2cl-lib:fstring-= c4 "QL")
                (f2cl-lib:fstring-= c4 "HR")
                (f2cl-lib:fstring-= c4 "TR")
                (f2cl-lib:fstring-= c4 "BR"))
            (setf nbmin 2)))))))

```



```

                (f2cl-lib:fstring-= c4 "BR"))
            (setf nbmin 2))))))
    (setf ilaenv nbmin)
    (go end_label)
label300
    (setf nx 0)
    (cond
      ((f2cl-lib:fstring-= c2 "GE")
        (cond
          ((or (f2cl-lib:fstring-= c3 "QRF")
                (f2cl-lib:fstring-= c3 "RQF")
                (f2cl-lib:fstring-= c3 "LQF")
                (f2cl-lib:fstring-= c3 "QLF"))
            (cond
              (sname
               (setf nx 128))
              (t
               (setf nx 128))))
            ((f2cl-lib:fstring-= c3 "HRD")
              (cond
                (sname
                 (setf nx 128))
                (t
                 (setf nx 128))))
            ((f2cl-lib:fstring-= c3 "BRD")
              (cond
                (sname
                 (setf nx 128))
                (t
                 (setf nx 128))))))
        ((f2cl-lib:fstring-= c2 "SY")
          (cond
            ((and sname (f2cl-lib:fstring-= c3 "TRD"))
              (setf nx 32))))
          ((and cname (f2cl-lib:fstring-= c2 "HE"))
            (cond
              ((f2cl-lib:fstring-= c3 "TRD")
                (setf nx 32))))
          ((and sname (f2cl-lib:fstring-= c2 "OR"))
            (cond
              ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
                (cond
                  ((or (f2cl-lib:fstring-= c4 "QR")
                        (f2cl-lib:fstring-= c4 "RQ")
                        (f2cl-lib:fstring-= c4 "LQ")
                        (f2cl-lib:fstring-= c4 "QL")
                        (f2cl-lib:fstring-= c4 "HR")
                        (f2cl-lib:fstring-= c4 "TR")
                        (f2cl-lib:fstring-= c4 "BR"))
                    (setf nx 128))))))

```

```

((and cname (f2cl-lib:fstring-= c2 "UN"))
(cond
  ((f2cl-lib:fstring-= (f2cl-lib:fref-string c3 (1 1)) "G")
   (cond
    ((or (f2cl-lib:fstring-= c4 "QR")
         (f2cl-lib:fstring-= c4 "RQ")
         (f2cl-lib:fstring-= c4 "LQ")
         (f2cl-lib:fstring-= c4 "QL")
         (f2cl-lib:fstring-= c4 "HR")
         (f2cl-lib:fstring-= c4 "TR")
         (f2cl-lib:fstring-= c4 "BR"))
     (setf nx 128))))))
  (setf ilaenv nx)
  (go end_label)
label400
  (setf ilaenv 6)
  (go end_label)
label500
  (setf ilaenv 2)
  (go end_label)
label600
  (setf ilaenv
    (f2cl-lib:int
     (*
      (coerce (realpart
                (min (the fixnum n1) (the fixnum n2))) 'single-float)
              1.6f0)))
  (go end_label)
label700
  (setf ilaenv 1)
  (go end_label)
label800
  (setf ilaenv 50)
  (go end_label)
label900
  (setf ilaenv 25)
  (go end_label)
label1000
  (setf ilaenv 0)
  (cond
   ((= ilaenv 1)
    (setf ilaenv (ieeek 0 0.0f0 1.0f0))))
  (go end_label)
label1100
  (setf ilaenv 0)
  (cond
   ((= ilaenv 1)
    (setf ilaenv (ieeek 1 0.0f0 1.0f0))))
end_label
  (return (values ilaenv nil nil nil nil nil nil))))

```

ilazlc LAPACK

— ilazlc.input —

```
)set break resume
)sys rm -f ilazlc.output
)spool ilazlc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— ilazlc.help —

```
=====
ilazlc examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
INTEGER FUNCTION ILAZLC( M, N, A, LDA )
```

```
.. Scalar Arguments ..
```

```
INTEGER          M, N, LDA
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       A( LDA, * )
```

```
..
```

Purpose:

=====

ILAZLC scans A for its last non-zero column.

Arguments:

=====

[in] M

M is INTEGER

The number of rows of the matrix A.

[in] N

N is INTEGER

The number of columns of the matrix A.

[in] A

A is COMPLEX*16 array, dimension (LDA,N)

The m by n matrix A.

[in] LDA

LDA is INTEGER

The leading dimension of the array A. LDA \geq max(1,M).

Authors:

=====

Univ. of Tennessee

Univ. of California Berkeley

Univ. of Colorado Denver

NAG Ltd.

November 2011

— ilazlc.f —

```

* =====
*      INTEGER FUNCTION ILAZLC( M, N, A, LDA )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER                M, N, LDA

```

```

*      ..
*      .. Array Arguments ..
*      COMPLEX*16      A( LDA, * )
*      ..
*
*      =====
*
*      .. Parameters ..
*      COMPLEX*16      ZERO
*      PARAMETER ( ZERO = (0.0D+0, 0.0D+0) )
*      ..
*      .. Local Scalars ..
*      INTEGER I
*      ..
*      .. Executable Statements ..
*
*      Quick test for the common case where one corner is non-zero.
*      IF( N.EQ.0 ) THEN
*          ILAZLC = N
*      ELSE IF( A(1, N).NE.ZERO .OR. A(M, N).NE.ZERO ) THEN
*          ILAZLC = N
*      ELSE
*
*      Now scan each column from the end, returning with the first non-zero.
*          DO ILAZLC = N, 1, -1
*              DO I = 1, M
*                  IF( A(I, ILAZLC).NE.ZERO ) RETURN
*              END DO
*          END DO
*      END IF
*      RETURN
*      END

```

— LAPACK ilazlc —

```

(let* ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) zero) (ignorable zero))
  (defun ilazlc (m n a lda)
    (declare (type (f2cl-lib:integer4) lda n m)
      (type (array f2cl-lib:complex16 (*)) a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%))
      (prog
        ((i 0) (ilazlc 0)) (declare (type (f2cl-lib:integer4) ilazlc i))
        (cond ((= n 0) (setf ilazlc n))
          ((or (/= (f2cl-lib:fref a (1 n) ((1 lda) (1 *))) zero)
              (/= (f2cl-lib:fref a (m n) ((1 lda) (1 *))) zero))

```

```

      (setf ilazlc n))
    (t
     (f2cl-lib:fdo (ilazlc n (f2cl-lib:int-add ilazlc (f2cl-lib:int-sub 1)))
      ((>
        ilazlc 1)
       nil)
      (tagbody
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i m) nil)
        (tagbody
         (if
          (/= (f2cl-lib:fref a-%data% (i ilazlc)
                           ((1 lda) (1 *)) a-%offset%) zero)
          (go end_label))
          label100001))
         label100000))))
      (go end_label) end_label (return (values ilazlc nil nil nil nil))))))

```

ilazlr LAPACK

— ilazlr.input —

```

)set break resume
)sys rm -f ilazlr.output
)spool ilazlr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ilazlr.help —

```

=====
ilazlr examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```
INTEGER FUNCTION ILAZLR( M, N, A, LDA )
```

```
.. Scalar Arguments ..
INTEGER                M, N, LDA
..
.. Array Arguments ..
COMPLEX*16             A( LDA, * )
..
```

Purpose:

=====

ILAZLR scans A for its last non-zero row.

Arguments:

=====

[in] M

M is INTEGER
 The number of rows of the matrix A.

[in] N

N is INTEGER
 The number of columns of the matrix A.

[in] A

A is COMPLEX*16 array, dimension (LDA,N)
 The m by n matrix A.

[in] LDA

LDA is INTEGER
 The leading dimension of the array A. LDA >= max(1,M).

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— ilazlr.f —

```

* =====
*      INTEGER FUNCTION ILAZLR( M, N, A, LDA )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,      --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          M, N, LDA
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * )
*
*      ..
*
* =====
*
*      .. Parameters ..
*      COMPLEX*16       ZERO
*      PARAMETER ( ZERO = (0.0D+0, 0.0D+0) )
*
*      ..
*      .. Local Scalars ..
*      INTEGER I, J
*
*      ..
*      .. Executable Statements ..
*
*      Quick test for the common case where one corner is non-zero.
*      IF( M.EQ.0 ) THEN
*          ILAZLR = M
*      ELSE IF( A(M, 1).NE.ZERO .OR. A(M, N).NE.ZERO ) THEN
*          ILAZLR = M
*      ELSE
*
*      Scan up each column tracking the last zero row seen.
*          ILAZLR = 0
*          DO J = 1, N
*              I=M
*              DO WHILE ((A(I,J).NE.ZERO).AND.(I.GE.1))
*                  I=I-1
*                  IF (I.EQ.0) THEN
*                      EXIT
*                  END IF

```



```

        ENDDO
        ILAZLR = MAX( ILAZLR, I )
    END DO
END IF
RETURN
END

```

— LAPACK ilazlr —

```

(let* ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) zero) (ignorable zero))
  (defun ilazlr (m n a lda)
    (declare (type (f2cl-lib:integer4) lda n m)
      (type (array f2cl-lib:complex16 (*)) a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%))
      (prog
        ((i 0) (j 0) (ilazlr 0))
        (declare (type (f2cl-lib:integer4) ilazlr j i))
        (cond ((= m 0) (setf ilazlr m))
          ((or (/= (f2cl-lib:fref a (m 1) ((1 lda) (1 *))) zero)
              (/= (f2cl-lib:fref a (m n) ((1 lda) (1 *))) zero))
            (setf ilazlr m))
          (t (setf ilazlr 0)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
              (> j n) nil)
            (tagbody (setf i m)
              label100001
              (if
                (not
                  (and (/= (f2cl-lib:fref a-%data% (i j)
                    ((1 lda) (1 *)) a-%offset%) zero)
                    (>= i 1)))
                (go label100002))
              (setf i (f2cl-lib:int-sub i 1))
              (cond ((= i 0) (go f2cl-lib::exit)))
              (go label100001) label100002
              (setf ilazlr
                (max (the f2cl-lib:integer4 ilazlr)
                  (the f2cl-lib:integer4 i)))
              label100000))))))
    (go end_label) end_label (return (values ilazlr nil nil nil nil))))))}

```

zgebak LAPACK

— zgebak.input —

```
)set break resume
)sys rm -f zgebak.output
)spool zgebak.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zgebak.help —

```
=====
zgebak examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
SUBROUTINE ZGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV,
                  INFO )
```

```
.. Scalar Arguments ..
CHARACTER            JOB, SIDE
INTEGER              IHI, ILO, INFO, LDV, M, N
..
.. Array Arguments ..
DOUBLE PRECISION     SCALE( * )
COMPLEX*16           V( LDV, * )
..
```

Purpose:
 =====

ZGEBAL forms the right or left eigenvectors of a complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by ZGEBAL.

Arguments:
=====

[in] JOB

JOB is CHARACTER*1
Specifies the type of backward transformation required:
= 'N', do nothing, return immediately;
= 'P', do backward transformation for permutation only;
= 'S', do backward transformation for scaling only;
= 'B', do backward transformations for both permutation and scaling.
JOB must be the same as the argument JOB supplied to ZGEBAL.

[in] SIDE

SIDE is CHARACTER*1
= 'R': V contains right eigenvectors;
= 'L': V contains left eigenvectors.

[in] N

N is INTEGER
The number of rows of the matrix V. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER
The integers ILO and IHI determined by ZGEBAL.
 $1 \leq \text{ILO} \leq \text{IHI} \leq N$, if $N > 0$; $\text{ILO}=1$ and $\text{IHI}=0$, if $N=0$.

[in] SCALE

SCALE is DOUBLE PRECISION array, dimension (N)
Details of the permutation and scaling factors, as returned by ZGEBAL.

[in] M

M is INTEGER

The number of columns of the matrix V. $M \geq 0$.

[in,out] V

V is COMPLEX*16 array, dimension (LDV,M)
 On entry, the matrix of right or left eigenvectors to be transformed, as returned by ZHSEIN or ZTREVC.
 On exit, V is overwritten by the transformed eigenvectors.

[in] LDV

LDV is INTEGER
 The leading dimension of the array V. $LDV \geq \max(1,N)$.

[out] INFO

INFO is INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zgebak.f —

```
* =====
*      SUBROUTINE ZGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV,
*      $                  INFO )
*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          JOB, SIDE
*      INTEGER            IHI, ILO, INFO, LDV, M, N
*      ..
*      .. Array Arguments ..
```

```

      DOUBLE PRECISION  SCALE( * )
      COMPLEX*16        V( LDV, * )
*
* ..
*
* =====
*
* .. Parameters ..
      DOUBLE PRECISION  ONE
      PARAMETER          ( ONE = 1.0D+0 )
*
* ..
* .. Local Scalars ..
      LOGICAL            LEFTV, RIGHTV
      INTEGER            I, II, K
      DOUBLE PRECISION  S
*
* ..
* .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*
* ..
* .. External Subroutines ..
      EXTERNAL           XERBLA, ZDSCAL, ZSWAP
*
* ..
* .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*
* ..
* .. Executable Statements ..
*
* Decode and Test the input parameters
*
*
      RIGHTV = LSAME( SIDE, 'R' )
      LEFTV = LSAME( SIDE, 'L' )
*
*
      INFO = 0
      IF( .NOT.LSAME( JOB, 'N' ) .AND. .NOT.LSAME( JOB, 'P' ) .AND.
$      .NOT.LSAME( JOB, 'S' ) .AND. .NOT.LSAME( JOB, 'B' ) ) THEN
         INFO = -1
      ELSE IF( .NOT.RIGHTV .AND. .NOT.LEFTV ) THEN
         INFO = -2
      ELSE IF( N.LT.0 ) THEN
         INFO = -3
      ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
         INFO = -4
      ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
         INFO = -5
      ELSE IF( M.LT.0 ) THEN
         INFO = -7
      ELSE IF( LDV.LT.MAX( 1, N ) ) THEN
         INFO = -9
      END IF
      IF( INFO.NE.0 ) THEN

```

```

        CALL XERBLA( 'ZGEBAK', -INFO )
        RETURN
    END IF
*
*   Quick return if possible
*
    IF( N.EQ.0 )
    $   RETURN
    IF( M.EQ.0 )
    $   RETURN
    IF( LSAME( JOB, 'N' ) )
    $   RETURN
*
    IF( ILO.EQ.IHI )
    $   GO TO 30
*
*   Backward balance
*
    IF( LSAME( JOB, 'S' ) .OR. LSAME( JOB, 'B' ) ) THEN
*
        IF( RIGHTV ) THEN
            DO 10 I = ILO, IHI
                S = SCALE( I )
                CALL ZDSCAL( M, S, V( I, 1 ), LDV )
10          CONTINUE
        END IF
*
        IF( LEFTV ) THEN
            DO 20 I = ILO, IHI
                S = ONE / SCALE( I )
                CALL ZDSCAL( M, S, V( I, 1 ), LDV )
20          CONTINUE
        END IF
*
    END IF
*
*   Backward permutation
*
    For I = ILO-1 step -1 until 1,
    $   IHI+1 step 1 until N do --
*
30  CONTINUE
    IF( LSAME( JOB, 'P' ) .OR. LSAME( JOB, 'B' ) ) THEN
        IF( RIGHTV ) THEN
            DO 40 II = 1, N
                I = II
                IF( I.GE.ILO .AND. I.LE.IHI )
                $   GO TO 40
                IF( I.LT.ILO )
                $   I = ILO - II

```

```

                K = SCALE( I )
                IF( K.EQ.I )
$                 GO TO 40
                CALL ZSWAP( M, V( I, 1 ), LDV, V( K, 1 ), LDV )
40             CONTINUE
            END IF
*
            IF( LEFTV ) THEN
                DO 50 II = 1, N
                    I = II
                    IF( I.GE.ILO .AND. I.LE.IHI )
$                     GO TO 50
                    IF( I.LT.ILO )
$                     I = ILO - II
                    K = SCALE( I )
                    IF( K.EQ.I )
$                     GO TO 50
                    CALL ZSWAP( M, V( I, 1 ), LDV, V( K, 1 ), LDV )
50             CONTINUE
            END IF
        END IF
*
        RETURN
*
*   End of ZGEBAK
*
END

```

— LAPACK zgebak —

```

(let* ((one 1.0d0))
  (declare (type (double-float 1.0d0 1.0d0) one) (ignorable one))
  (defun zgebak (job side n ilo ihi scale m v ldv info)
    (declare (type (simple-array character (*)) side job)
      (type (f2cl-lib:integer4) info ldv m ihi ilo n)
      (type (array double-float (*)) scale)
      (type (array f2cl-lib:complex16 (*)) v))
    (f2cl-lib:with-multi-array-data
      ((v f2cl-lib:complex16 v-%data% v-%offset%)
       (scale double-float scale-%data% scale-%offset%)
       (job character job-%data% job-%offset%)
       (side character side-%data% side-%offset%))
      (prog
        ((s 0.0d0) (i 0) (ii 0) (k 0) (leftv nil) (rightv nil))
        (declare (type (double-float) s) (type (f2cl-lib:integer4) k ii i)
          (type f2cl-lib:logical rightv leftv))

```

```

(setf rightv
  (multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
    (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
(setf leftv
  (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
    (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
(setf info 0)
(cond
  ((and
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "N")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "P")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "S")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "B")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))))
    (setf info -1))
  ((and (not rightv) (not leftv))
    (setf info -2)) ((< n 0) (setf info -3))
  ((or (< ilo 1)
    (> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))
    (setf info -4))
  ((or (< ihi (min (the f2cl-lib:integer4 ilo)
    (the f2cl-lib:integer4 n)))
    (> ihi n))
    (setf info -5))
  ((< m 0) (setf info -7))
  ((< ldv (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
    (setf info -9)))
(cond ((/= info 0)
  (xerbla "ZGEBAK" (f2cl-lib:int-sub info)) (go end_label)))
(if (= n 0) (go end_label)) (if (= m 0) (go end_label))
(if
  (multiple-value-bind (ret-val var-0 var-1) (lsame job "N")
    (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
  (go end_label))
(if (= ilo ihi) (go label30))
(cond
  ((or
    (multiple-value-bind (ret-val var-0 var-1) (lsame job "S")
      (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
    (multiple-value-bind (ret-val var-0 var-1) (lsame job "B")
      (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (cond
      (rightv

```



```

(f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
  (> i ihi) nil)
  (tagbody
    (setf s
      (f2cl-lib:fref scale-%data% (i) ((1 *))
        scale-%offset%))
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (zdscal m s
        (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
          (i 1) ((1 ldv) (1 *))
          v-%offset%)
        ldv)
      (declare (ignore var-2))
      (when var-0 (setf m var-0))
      (when var-1 (setf s var-1))
      (when var-3 (setf ldv var-3)))
    label10))))
(cond
  (leftv
    (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
      (> i ihi) nil)
      (tagbody
        (setf s
          (/ one
            (f2cl-lib:fref scale-%data% (i) ((1 *))
              scale-%offset%)))
        (multiple-value-bind (var-0 var-1 var-2 var-3)
          (zdscal m s
            (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
              (i 1) ((1 ldv) (1 *))
              v-%offset%)
            ldv)
          (declare (ignore var-2))
          (when var-0 (setf m var-0))
          (when var-1 (setf s var-1))
          (when var-3 (setf ldv var-3)))
          label20))))))
label30
(cond
  ((or
    (multiple-value-bind (ret-val var-0 var-1) (lsame job "P")
      (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
    (multiple-value-bind (ret-val var-0 var-1) (lsame job "B")
      (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (cond
      (rightv
        (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
          (> ii n) nil)
          (tagbody
            (setf i ii)

```

```

(if (and (>= i ilo) (<= i ihi)) (go label40))
(if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
(setf k
  (f2cl-lib:int
    (f2cl-lib:fref scale-%data% (i) ((1 *))
      scale-%offset%)))
(if (= k i) (go label40))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (zswap m
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
      (i 1) ((1 ldv) (1 *))
      v-%offset%)
    ldv
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
      (k 1) ((1 ldv) (1 *))
      v-%offset%)
    ldv)
  (declare (ignore var-1 var-3))
  (when var-0 (setf m var-0))
  (when var-2 (setf ldv var-2))
  (when var-4 (setf ldv var-4)))
label40)))

(cond
  (leftv
    (f2cl-lib:fdo (ii 1 (f2cl-lib:int-add ii 1))
      ((> ii n) nil)
      (tagbody
        (setf i ii)
        (if (and (>= i ilo) (<= i ihi)) (go label50))
        (if (< i ilo) (setf i (f2cl-lib:int-sub ilo ii)))
        (setf k
          (f2cl-lib:int
            (f2cl-lib:fref scale-%data% (i) ((1 *))
              scale-%offset%)))
        (if (= k i) (go label50))
        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
          (zswap m
            (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
              (i 1) ((1 ldv) (1 *))
              v-%offset%)
            ldv
            (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
              (k 1) ((1 ldv) (1 *))
              v-%offset%)
            ldv)
          (declare (ignore var-1 var-3))
          (when var-0 (setf m var-0))
          (when var-2 (setf ldv var-2))
          (when var-4 (setf ldv var-4)))
        label50))))))

```

```
(go end_label) end_label
(return (values job side nil nil nil nil m nil ldv info))))))
```

zgebal LAPACK

— zgebal.input —

```
)set break resume
)sys rm -f zgebal.output
)spool zgebal.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zgebal.help —

```
=====
zgebal examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
```

```
.. Scalar Arguments ..
CHARACTER          JOB
INTEGER            IHI, ILO, INFO, LDA, N
..
.. Array Arguments ..
DOUBLE PRECISION   SCALE( * )
COMPLEX*16         A( LDA, * )
```

..

Purpose:

=====

ZGEBAL balances a general complex matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ILO-1 and last IHI+1 to N elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ILO to IHI to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Arguments:

=====

[in] JOB

JOB is CHARACTER*1

Specifies the operations to be performed on A:

= 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0
 for i = 1,...,N;
 = 'P': permute only;
 = 'S': scale only;
 = 'B': both permute and scale.

[in] N

N is INTEGER

The order of the matrix A. N >= 0.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)

On entry, the input matrix A.

On exit, A is overwritten by the balanced matrix.

If JOB = 'N', A is not referenced.

See Further Details.

[in] LDA

LDA is INTEGER

The leading dimension of the array A. LDA >= max(1,N).

[out] ILO

[out] IHI

ILO and IHI are set to INTEGER such that on exit
 $A(i,j) = 0$ if $i > j$ and $j = 1, \dots, ILO-1$ or $i = IHI+1, \dots, N$.
 If JOB = 'N' or 'S', ILO = 1 and IHI = N.

[out] SCALE

SCALE is DOUBLE PRECISION array, dimension (N)
 Details of the permutations and scaling factors applied to
 A. If P(j) is the index of the row and column interchanged
 with row and column j and D(j) is the scaling factor
 applied to row and column j, then
 $SCALE(j) = P(j)$ for $j = 1, \dots, ILO-1$
 $= D(j)$ for $j = ILO, \dots, IHI$
 $= P(j)$ for $j = IHI+1, \dots, N$.
 The order in which the interchanges are made is N to IHI+1,
 then 1 to ILO-1.

[out] INFO

INFO is INTEGER
 = 0: successful exit.
 < 0: if INFO = -i, the i-th argument had an illegal value.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:

=====

The permutations consist of row and column interchanges which put
 the matrix in the form

$$P A P = \begin{pmatrix} T1 & X & Y \\ 0 & B & Z \\ 0 & 0 & T2 \end{pmatrix}$$

where T1 and T2 are upper triangular matrices whose eigenvalues lie
 along the diagonal. The column indices ILO and IHI mark the starting
 and ending columns of the submatrix B. Balancing consists of applying
 a diagonal similarity transformation $\text{inv}(D) * B * D$ to make the
 1-norms of each row of B and its corresponding column nearly equal.

The output matrix is

$$\begin{pmatrix} T1 & X*D & Y \\ 0 & \text{inv}(D)*B*D & \text{inv}(D)*Z \\ 0 & 0 & T2 \end{pmatrix}.$$

Information about the permutations P and the diagonal matrix D is returned in the vector SCALE.

This subroutine is based on the EISPACK routine CBAL.

Modified by Tzu-Yi Chen, Computer Science Division, University of California at Berkeley, USA

— zgebal.f —

```
* =====
*      SUBROUTINE ZGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          JOB
*      INTEGER            IHI, ILO, INFO, LDA, N
*
*      ..
*      .. Array Arguments ..
*      DOUBLE PRECISION   SCALE( * )
*      COMPLEX*16         A( LDA, * )
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION   ZERO, ONE
*      PARAMETER          ( ZERO = 0.0D+0, ONE = 1.0D+0 )
*      DOUBLE PRECISION   SCLFAC
*      PARAMETER          ( SCLFAC = 2.0D+0 )
*      DOUBLE PRECISION   FACTOR
*      PARAMETER          ( FACTOR = 0.95D+0 )
*      ..
*      .. Local Scalars ..
*      LOGICAL            NOCONV
*      INTEGER            I, ICA, IEXC, IRA, J, K, L, M
```

```

      DOUBLE PRECISION  C, CA, F, G, R, RA, S, SFMAX1, SFMAX2, SFMIN1,
$      SFMIN2
      COMPLEX*16        CDUM
*      ..
*      .. External Functions ..
      LOGICAL           DISNAN, LSAME
      INTEGER           IZAMAX
      DOUBLE PRECISION  DLAMCH
      EXTERNAL          DISNAN, LSAME, IZAMAX, DLAMCH
*      ..
*      .. External Subroutines ..
      EXTERNAL          XERBLA, ZDSCAL, ZSWAP
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC         ABS, DBLE, DIMAG, MAX, MIN
*      ..
*      .. Statement Functions ..
      DOUBLE PRECISION  CABS1
*      ..
*      .. Statement Function definitions ..
      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters
*
      INFO = 0
      IF( .NOT.LSAME( JOB, 'N' ) .AND. .NOT.LSAME( JOB, 'P' ) .AND.
$      .NOT.LSAME( JOB, 'S' ) .AND. .NOT.LSAME( JOB, 'B' ) ) THEN
          INFO = -1
      ELSE IF( N.LT.0 ) THEN
          INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZGEBAL', -INFO )
          RETURN
      END IF
*
      K = 1
      L = N
*
      IF( N.EQ.0 )
$      GO TO 210
*
      IF( LSAME( JOB, 'N' ) ) THEN
          DO 10 I = 1, N
              SCALE( I ) = ONE
10      CONTINUE

```

```

        GO TO 210
      END IF
*
      IF( LSAME( JOB, 'S' ) )
$      GO TO 120
*
*      Permutation to isolate eigenvalues if possible
*
      GO TO 50
*
*      Row and column exchange.
*
20  CONTINUE
      SCALE( M ) = J
      IF( J.EQ.M )
$      GO TO 30
*
      CALL ZSWAP( L, A( 1, J ), 1, A( 1, M ), 1 )
      CALL ZSWAP( N-K+1, A( J, K ), LDA, A( M, K ), LDA )
*
30  CONTINUE
      GO TO ( 40, 80 )IEXC
*
*      Search for rows isolating an eigenvalue and push them down.
*
40  CONTINUE
      IF( L.EQ.1 )
$      GO TO 210
      L = L - 1
*
50  CONTINUE
      DO 70 J = L, 1, -1
*
          DO 60 I = 1, L
              IF( I.EQ.J )
$                  GO TO 60
                  IF( DBLE( A( J, I ) ).NE.ZERO .OR. DIMAG( A( J, I ) ).NE.
$                      ZERO )GO TO 70
60      CONTINUE
*
          M = L
          IEXC = 1
          GO TO 20
70  CONTINUE
*
      GO TO 90
*
*      Search for columns isolating an eigenvalue and push them left.
*
80  CONTINUE

```



```

      K = K + 1
*
90  CONTINUE
      DO 110 J = K, L
*
          DO 100 I = K, L
              IF( I.EQ.J )
$                  GO TO 100
              IF( DBLE( A( I, J ) ).NE.ZERO .OR. DIMAG( A( I, J ) ).NE.
$                  ZERO )GO TO 110
100  CONTINUE
*
          M = K
          IEXC = 2
          GO TO 20
110  CONTINUE
*
120  CONTINUE
      DO 130 I = K, L
          SCALE( I ) = ONE
130  CONTINUE
*
      IF( LSAME( JOB, 'P' ) )
$          GO TO 210
*
*      Balance the submatrix in rows K to L.
*
*      Iterative loop for norm reduction
*
      SFMIN1 = DLAMCH( 'S' ) / DLAMCH( 'P' )
      SFMAX1 = ONE / SFMIN1
      SFMIN2 = SFMIN1*SCLFAC
      SFMAX2 = ONE / SFMIN2
140  CONTINUE
      NOCONV = .FALSE.
*
      DO 200 I = K, L
          C = ZERO
          R = ZERO
*
          DO 150 J = K, L
              IF( J.EQ.I )
$                  GO TO 150
              C = C + CABS1( A( J, I ) )
              R = R + CABS1( A( I, J ) )
150  CONTINUE
          ICA = IZAMAX( L, A( 1, I ), 1 )
          CA = ABS( A( ICA, I ) )
          IRA = IZAMAX( N-K+1, A( I, K ), LDA )
          RA = ABS( A( I, IRA+K-1 ) )

```

```

*
*      Guard against zero C or R due to underflow.
*
      IF( C.EQ.ZERO .OR. R.EQ.ZERO )
$      GO TO 200
      G = R / SCLFAC
      F = ONE
      S = C + R
160  CONTINUE
      IF( C.GE.G .OR. MAX( F, C, CA ).GE.SFMAX2 .OR.
$      MIN( R, G, RA ).LE.SFMIN2 )GO TO 170
      IF( DISNAN( C+F+CA+R+G+RA ) ) THEN
*
*      Exit if NaN to avoid infinite loop
*
      INFO = -3
      CALL XERBLA( 'ZGEBAL', -INFO )
      RETURN
      END IF
      F = F*SCLFAC
      C = C*SCLFAC
      CA = CA*SCLFAC
      R = R / SCLFAC
      G = G / SCLFAC
      RA = RA / SCLFAC
      GO TO 160
*
170  CONTINUE
      G = C / SCLFAC
180  CONTINUE
      IF( G.LT.R .OR. MAX( R, RA ).GE.SFMAX2 .OR.
$      MIN( F, C, G, CA ).LE.SFMIN2 )GO TO 190
      F = F / SCLFAC
      C = C / SCLFAC
      G = G / SCLFAC
      CA = CA / SCLFAC
      R = R*SCLFAC
      RA = RA*SCLFAC
      GO TO 180
*
*      Now balance.
*
190  CONTINUE
      IF( ( C+R ).GE.FACTOR*S )
$      GO TO 200
      IF( F.LT.ONE .AND. SCALE( I ).LT.ONE ) THEN
          IF( F*SCALE( I ).LE.SFMIN1 )
$      GO TO 200
      END IF
      IF( F.GT.ONE .AND. SCALE( I ).GT.ONE ) THEN

```

```

        IF( SCALE( I ).GE.SFMAX1 / F )
$      GO TO 200
      END IF
      G = ONE / F
      SCALE( I ) = SCALE( I )*F
      NOCONV = .TRUE.
*
      CALL ZDSCAL( N-K+1, G, A( I, K ), LDA )
      CALL ZDSCAL( L, F, A( 1, I ), 1 )
*
200 CONTINUE
*
      IF( NOCONV )
$      GO TO 140
*
210 CONTINUE
      ILO = K
      IHI = L
*
      RETURN
*
*      End of ZGEBAL
*
      END

```

— LAPACK zgebal —

```

(let* ((zero 0.0d0) (one 1.0d0) (sclfac 2.0d0) (factor 0.95d0))
(declare (type (double-float 0.0d0 0.0d0) zero)
(type (double-float 1.0d0 1.0d0) one)
(type (double-float 2.0d0 2.0d0) sclfac)
(type (double-float 0.95d0 0.95d0) factor)
(ignorable zero one sclfac factor))
(defun zgebal (job n a lda ilo ihi scale info)
(declare (type (simple-array character (*)) job)
(type (f2cl-lib:integer4) info ihi ilo lda n)
(type (array f2cl-lib:complex16 (*)) a)
(type (array double-float (*)) scale))
(f2cl-lib:with-multi-array-data
((scale double-float scale-%data%
scale-%offset%)
(a f2cl-lib:complex16 a-%data% a-%offset%)
(job character job-%data% job-%offset%))
(labels
((cabs1 (cdum)
(+ (abs (f2cl-lib:dbler cdum)) (abs (f2cl-lib:dimag cdum))))))

```

```

(declare
  (ftype (function (f2cl-lib:complex16)
    (values double-float &rest t)) cabs1))
(prog
  ((cdum #C(0.0d0 0.0d0)) (c 0.0d0) (ca 0.0d0) (f 0.0d0)
    (g 0.0d0) (r 0.0d0)
  (ra 0.0d0) (s 0.0d0) (sfxmax1 0.0d0) (sfxmax2 0.0d0) (sfmin1 0.0d0)
  (sfmin2 0.0d0) (i 0) (ica 0) (iexc 0) (ira 0) (j 0) (k 0) (l 0) (m 0)
  (noconv nil))
(declare (type (f2cl-lib:complex16) cdum)
  (type (double-float) sfmin2 sfmin1 sfxmax2 sfxmax1 s ra r g f ca c)
  (type (f2cl-lib:integer4) m l k j ira iexc ica i)
  (type f2cl-lib:logical noconv))
(setf info 0)
(cond
  ((and
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "N")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "P")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "S")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame job "B")
        (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))))
  (setf info -1))
  ((< n 0) (setf info -2))
  ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
    (setf info -4)))
(cond ((/= info 0)
  (xerbla "ZGEBAL" (f2cl-lib:int-sub info))
  (go end_label)))
(setf k 1) (setf l n) (if (= n 0) (go label210))
(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame job "N")
    (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref scale-%data% (i) ((1 *))
        scale-%offset%) one)
      label10))
  (go label210)))
(if
  (multiple-value-bind (ret-val var-0 var-1) (lsame job "S")
    (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
  (go label120))

```

```

(go label150) label120
(setf (f2cl-lib:fref scale-%data% (m) ((1 *)) scale-%offset%)
  (coerce (the f2cl-lib:integer4 j) 'double-float))
(if (= j m) (go label130))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (zswap 1
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      (1 j) ((1 lda) (1 *)))
    a-%offset%)
  1
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    (1 m) ((1 lda) (1 *)))
  a-%offset%)
  1)
(declare (ignore var-1 var-2 var-3 var-4))
(when var-0 (setf 1 var-0)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (zswap (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      (j k) ((1 lda) (1 *)))
    a-%offset%)
  lda
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    (m k) ((1 lda) (1 *)))
  a-%offset%)
  lda)
(declare (ignore var-0 var-1 var-3)) (when var-2 (setf lda var-2))
(when var-4 (setf lda var-4)))
label130 (f2cl-lib:computed-goto (label140 label180) iexc) label140
(if (= 1 1) (go label210)) (setf 1 (f2cl-lib:int-sub 1 1)) label150
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  (> j 1) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i 1) nil)
    (tagbody
      (if (= i j) (go label160))
      (if
        (or
          (/=
            (f2cl-lib:dble
              (f2cl-lib:fref a-%data% (j i) ((1 lda) (1 *)))
              a-%offset%))
            zero)
          (/=
            (f2cl-lib:dimag
              (f2cl-lib:fref a-%data% (j i) ((1 lda) (1 *)))
              a-%offset%))
            zero))
        (go label170)))

```

```

        label160))
        (setf m 1) (setf iexc 1) (go label20) label170))
(go label190) label180 (setf k (f2cl-lib:int-add k 1)) label190
(f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
  (> j 1) nil)
  (tagbody
    (f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
      (> i 1) nil)
      (tagbody
        (if (= i j) (go label100))
        (if
          (or
            (/=
              (f2cl-lib:dbple
                (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *))
                a-%offset%))
              zero)
            (/=
              (f2cl-lib:dimag
                (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *))
                a-%offset%))
              zero))
          (go label110))
          label100))
        (setf m k) (setf iexc 2) (go label20) label110))
label120
(f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
  (> i 1) nil)
  (tagbody
    (setf (f2cl-lib:fref scale-%data% (i) ((1 *))
      scale-%offset%) one) label130)
  )
(if
  (multiple-value-bind (ret-val var-0 var-1) (lsame job "P")
    (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val)
  (go label210))
(setf sfmin1 (/ (dlamch "S") (dlamch "P")))
(setf sfmax1 (/ one sfmin1))
(setf sfmin2 (* sfmin1 sclfac)) (setf sfmax2 (/ one sfmin2)) label140
(setf noconv f2cl-lib:%false%)
(f2cl-lib:fdo (i k (f2cl-lib:int-add i 1))
  (> i 1) nil)
  (tagbody (setf c zero)
    (setf r zero)
    (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))
      (> j 1) nil)
      (tagbody
        (if (= j i) (go label150))
        (setf c
          (+ c

```

```

(cabs1
  (f2cl-lib:fref a-%data% (j i) ((1 lda) (1 *))
    a-%offset%)))
(setf r
  (+ r
    (cabs1
      (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *))
        a-%offset%))))
label1150))
(setf ica
  (multiple-value-bind (ret-val var-0 var-1 var-2)
    (izamax 1
      (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
        (1 i) ((1 lda) (1 *))
        a-%offset%)
      1)
    (declare (ignore var-1 var-2))
    (when var-0 (setf 1 var-0)) ret-val))
(setf ca
  (abs
    (f2cl-lib:fref a-%data% (ica i) ((1 lda) (1 *))
      a-%offset%)))
(setf ira
  (multiple-value-bind (ret-val var-0 var-1 var-2)
    (izamax (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1)
      (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
        (i k) ((1 lda) (1 *))
        a-%offset%)
      lda)
    (declare (ignore var-0 var-1))
    (when var-2 (setf lda var-2)) ret-val))
(setf ra
  (abs
    (f2cl-lib:fref a-%data%
      (i (f2cl-lib:int-sub (f2cl-lib:int-add ira k) 1))
      ((1 lda) (1 *)) a-%offset%)))
(if (or (= c zero) (= r zero)) (go label200))
(setf g (/ r sclfac))
(setf f one) (setf s (+ c r)) label1160
(if (or (>= c g)
      (>= (max f c ca) sfmax2)
      (<= (min r g ra) sfmin2))
  (go label1170))
(cond
  ((disnan (+ c f ca r g ra)) (setf info -3)
   (xerbla "ZGEBAL" (f2cl-lib:int-sub info))
   (go end_label)))
(setf f (* f sclfac))
(setf c (* c sclfac))
(setf ca (* ca sclfac))

```

```

(setf r (/ r sclfac))
(setf g (/ g sclfac))
(setf ra (/ ra sclfac))
(go label160) label170 (setf g (/ c sclfac)) label180
(if (or (< g r)
      (>= (max r ra) sfmax2)
      (<= (min f c g ca) sfmin2))
    (go label190))
(setf f (/ f sclfac))
(setf c (/ c sclfac))
(setf g (/ g sclfac))
(setf ca (/ ca sclfac))
(setf r (* r sclfac))
(setf ra (* ra sclfac))
(go label180)

label190

(if (>= (+ c r) (* factor s)) (go label200))
(cond
  ((and (< f one) (< (f2cl-lib:fref scale (i) ((1 *))) one))
    (if
      (<= (* f
            (f2cl-lib:fref scale-%data% (i) ((1 *))
                          scale-%offset%)) sfmin1)
      (go label200))))
(cond
  ((and (> f one) (> (f2cl-lib:fref scale (i) ((1 *))) one))
    (if
      (>= (f2cl-lib:fref scale-%data% (i) ((1 *))
                      scale-%offset%) (/ sfmax1 f))
      (go label200))))
(setf g (/ one f))
(setf (f2cl-lib:fref scale-%data% (i) ((1 *))
                    scale-%offset%)
      (* (f2cl-lib:fref scale-%data% (i) ((1 *))
          scale-%offset%) f))
(setf noconv f2cl-lib:true%)
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zdscal (f2cl-lib:int-add (f2cl-lib:int-sub n k) 1) g
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      (i k) ((1 lda) (1 *)))
    a-%offset%)
  lda)
(declare (ignore var-0 var-2)) (when var-1 (setf g var-1))
(when var-3 (setf lda var-3))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zdscal 1 f
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      (1 i) ((1 lda) (1 *)))
    a-%offset%)
  1)

```



```

        (declare (ignore var-2 var-3)) (when var-0 (setf l var-0))
        (when var-1 (setf f var-1)))
    label200))
(if noconv (go label140)) label210
  (setf ilo k) (setf ihi l) (go end_label)
end_label (return (values job nil nil lda ilo ihi nil info))))))

```

zggev LAPACK

— zggev.input —

```

)set break resume
)sys rm -f zggev.output
)spool zggev.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zggev.help —

```

=====
zggev examples
=====

```

```

=====
Man Page Details
=====

```

ZGEEV computes the eigenvalues and, optionally, the left and/or right eigenvectors for GE matrices

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```

SUBROUTINE ZGEEV( JOBV, N, A, LDA, W, VL, LDVL, VR, LDVR,

```

```

                                WORK, LWORK, RWORK, INFO )

.. Scalar Arguments ..
CHARACTER          JOBVL, JOBVR
INTEGER            INFO, LDA, LDVL, LDVR, LWORK, N
..
.. Array Arguments ..
DOUBLE PRECISION   RWORK( * )
COMPLEX*16         A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
$                  W( * ), WORK( * )
..

```

Purpose:

=====

ZGEEV computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector $v(j)$ of A satisfies

$$A * v(j) = \lambda(j) * v(j)$$

where $\lambda(j)$ is its eigenvalue.

The left eigenvector $u(j)$ of A satisfies

$$u(j)**H * A = \lambda(j) * u(j)**H$$

where $u(j)**H$ denotes the conjugate transpose of $u(j)$.

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Arguments:

=====

[in] JOBVL

JOBVL is CHARACTER*1

= 'N': left eigenvectors of A are not computed;

= 'V': left eigenvectors of A are computed.

[in] JOBVR

JOBVR is CHARACTER*1

= 'N': right eigenvectors of A are not computed;

= 'V': right eigenvectors of A are computed.

[in] N

N is INTEGER

The order of the matrix A. $N \geq 0$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)
On entry, the N-by-N matrix A.
On exit, A has been overwritten.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. LDA \geq max(1,N).

[out] W

W is COMPLEX*16 array, dimension (N)
W contains the computed eigenvalues.

[out] VL

VL is COMPLEX*16 array, dimension (LDVL,N)
If JOBVL = 'V', the left eigenvectors $u(j)$ are stored one after another in the columns of VL, in the same order as their eigenvalues.
If JOBVL = 'N', VL is not referenced.
 $u(j) = VL(:,j)$, the j-th column of VL.

[in] LDVL

LDVL is INTEGER
The leading dimension of the array VL. LDVL \geq 1; if JOBVL = 'V', LDVL \geq N.

[out] VR

VR is COMPLEX*16 array, dimension (LDVR,N)
If JOBVR = 'V', the right eigenvectors $v(j)$ are stored one after another in the columns of VR, in the same order as their eigenvalues.
If JOBVR = 'N', VR is not referenced.
 $v(j) = VR(:,j)$, the j-th column of VR.

[in] LDVR

LDVR is INTEGER
The leading dimension of the array VR. LDVR \geq 1; if JOBVR = 'V', LDVR \geq N.

[out] WORK

WORK is COMPLEX*16 array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER

The dimension of the array WORK. $LWORK \geq \max(1, 2*N)$.

For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

[out] RWORK

RWORK is DOUBLE PRECISION array, dimension (2*N)

[out] INFO

INFO is INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value.

> 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements and i+1:N of W contain eigenvalues which have converged.

Authors:

=====

Univ. of Tennessee

Univ. of California Berkeley

Univ. of Colorado Denver

NAG Ltd.

November 2011

— zgeev.f —

```
* =====
*      SUBROUTINE ZGEEV( JOBVL, JOBVR, N, A, LDA, W, VL, LDVL, VR, LDVR,
*      $                WORK, LWORK, RWORK, INFO )
*
*      -- LAPACK driver routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
```

```

*
*   .. Scalar Arguments ..
CHARACTER          JOBVL, JOBVR
INTEGER            INFO, LDA, LDVL, LDVR, LWORK, N
*
*   ..
*   .. Array Arguments ..
DOUBLE PRECISION   RWORK( * )
COMPLEX*16         A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
$                 W( * ), WORK( * )
*
*   ..
*
*   =====
*
*   .. Parameters ..
DOUBLE PRECISION   ZERO, ONE
PARAMETER          ( ZERO = 0.0DO, ONE = 1.0DO )
*
*   ..
*   .. Local Scalars ..
LOGICAL            LQUERY, SCALEA, WANTVL, WANTVR
CHARACTER          SIDE
INTEGER            HSWORK, I, IBAL, IERR, IHI, ILO, IRWORK, ITAU,
$                 IWRK, K, MAXWRK, MINWRK, NOUT
DOUBLE PRECISION   ANRM, BIGNUM, CSCALE, EPS, SCL, SMLNUM
COMPLEX*16         TMP
*
*   ..
*   .. Local Arrays ..
LOGICAL            SELECT( 1 )
DOUBLE PRECISION   DUM( 1 )
*
*   ..
*   .. External Subroutines ..
EXTERNAL           DLABAD, XERBLA, ZDSCAL, ZGEBAK, ZGEBAL, ZGEHRD,
$                 ZHSEQR, ZLACPY, ZLASCL, ZSCAL, ZTREVC, ZUNGHR
*
*   ..
*   .. External Functions ..
LOGICAL            LSAME
INTEGER            IDAMAX, ILAENV
DOUBLE PRECISION   DLAMCH, DZNRM2, ZLANGE
EXTERNAL           LSAME, IDAMAX, ILAENV, DLAMCH, DZNRM2, ZLANGE
*
*   ..
*   .. Intrinsic Functions ..
INTRINSIC          DBLE, DCMLPX, DCONJG, DIMAG, MAX, SQRT
*
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
*   INFO = 0
*   LQUERY = ( LWORK.EQ.-1 )
*   WANTVL = LSAME( JOBVL, 'V' )
*   WANTVR = LSAME( JOBVR, 'V' )

```

```

      IF( ( .NOT.WANTVL ) .AND. ( .NOT.LSAME( JOBVL, 'N' ) ) ) THEN
        INFO = -1
      ELSE IF( ( .NOT.WANTVR ) .AND. ( .NOT.LSAME( JOBVR, 'N' ) ) ) THEN
        INFO = -2
      ELSE IF( N.LT.0 ) THEN
        INFO = -3
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -5
      ELSE IF( LDVL.LT.1 .OR. ( WANTVL .AND. LDVL.LT.N ) ) THEN
        INFO = -8
      ELSE IF( LDVR.LT.1 .OR. ( WANTVR .AND. LDVR.LT.N ) ) THEN
        INFO = -10
      END IF

*
*   Compute workspace
*   (Note: Comments in the code beginning "Workspace:" describe the
*   minimal amount of workspace needed at that point in the code,
*   as well as the preferred amount for good performance.
*   CWorkspace refers to complex workspace, and RWorkspace to real
*   workspace. NB refers to the optimal block size for the
*   immediately following subroutine, as returned by ILAENV.
*   HSWORK refers to the workspace preferred by ZHSEQR, as
*   calculated below. HSWORK is computed assuming ILO=1 and IHI=N,
*   the worst case.)
*
      IF( INFO.EQ.0 ) THEN
        IF( N.EQ.0 ) THEN
          MINWRK = 1
          MAXWRK = 1
        ELSE
          MAXWRK = N + N*ILAENV( 1, 'ZGEHRD', ' ', N, 1, N, 0 )
          MINWRK = 2*N
          IF( WANTVL ) THEN
            MAXWRK = MAX( MAXWRK, N + ( N - 1 )*ILAENV( 1, 'ZUNGHR',
$              ' ', N, 1, N, -1 ) )
            CALL ZHSEQR( 'S', 'V', N, 1, N, A, LDA, W, VL, LDVL,
$              WORK, -1, INFO )
          ELSE IF( WANTVR ) THEN
            MAXWRK = MAX( MAXWRK, N + ( N - 1 )*ILAENV( 1, 'ZUNGHR',
$              ' ', N, 1, N, -1 ) )
            CALL ZHSEQR( 'S', 'V', N, 1, N, A, LDA, W, VR, LDVR,
$              WORK, -1, INFO )
          ELSE
            CALL ZHSEQR( 'E', 'N', N, 1, N, A, LDA, W, VR, LDVR,
$              WORK, -1, INFO )
          END IF
          HSWORK = WORK( 1 )
          MAXWRK = MAX( MAXWRK, HSWORK, MINWRK )
        END IF
        WORK( 1 ) = MAXWRK
      
```

```

*
      IF( LWORK.LT.MINWRK .AND. .NOT.LQUERY ) THEN
        INFO = -12
      END IF
    END IF
*
    IF( INFO.NE.0 ) THEN
      CALL XERBLA( 'ZGEEV ', -INFO )
      RETURN
    ELSE IF( LQUERY ) THEN
      RETURN
    END IF
*
*   Quick return if possible
*
    IF( N.EQ.0 )
$   RETURN
*
*   Get machine constants
*
    EPS = DLAMCH( 'P' )
    SMLNUM = DLAMCH( 'S' )
    BIGNUM = ONE / SMLNUM
    CALL DLABAD( SMLNUM, BIGNUM )
    SMLNUM = SQRT( SMLNUM ) / EPS
    BIGNUM = ONE / SMLNUM
*
*   Scale A if max element outside range [SMLNUM,BIGNUM]
*
    ANRM = ZLANGE( 'M', N, N, A, LDA, DUM )
    SCALEA = .FALSE.
    IF( ANRM.GT.ZERO .AND. ANRM.LT.SMLNUM ) THEN
      SCALEA = .TRUE.
      CSCALE = SMLNUM
    ELSE IF( ANRM.GT.BIGNUM ) THEN
      SCALEA = .TRUE.
      CSCALE = BIGNUM
    END IF
    IF( SCALEA )
$   CALL ZLASCL( 'G', 0, 0, ANRM, CSCALE, N, N, A, LDA, IERR )
*
*   Balance the matrix
*   (CWorkspace: none)
*   (RWorkspace: need N)
*
    IBAL = 1
    CALL ZGEBAL( 'B', N, A, LDA, ILO, IHI, RWORK( IBAL ), IERR )
*
*   Reduce to upper Hessenberg form
*   (CWorkspace: need 2*N, prefer N+N*NB)

```

```

*      (RWorkspace: none)
*
      ITAU = 1
      IWRK = ITAU + N
      CALL ZGEHRD( N, ILO, IHI, A, LDA, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
      IF( WANTVL ) THEN
*
*          Want left eigenvectors
*          Copy Householder vectors to VL
*
          SIDE = 'L'
          CALL ZLACPY( 'L', N, N, A, LDA, VL, LDVL )
*
*          Generate unitary matrix in VL
*          (CWorkspace: need 2*N-1, prefer N+(N-1)*NB)
*          (RWorkspace: none)
*
          CALL ZUNGHR( N, ILO, IHI, VL, LDVL, WORK( ITAU ), WORK( IWRK ),
$              LWORK-IWRK+1, IERR )
*
*          Perform QR iteration, accumulating Schur vectors in VL
*          (CWorkspace: need 1, prefer HSWORK (see comments) )
*          (RWorkspace: none)
*
          IWRK = ITAU
          CALL ZHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, W, VL, LDVL,
$              WORK( IWRK ), LWORK-IWRK+1, INFO )
*
          IF( WANTVR ) THEN
*
*              Want left and right eigenvectors
*              Copy Schur vectors to VR
*
              SIDE = 'B'
              CALL ZLACPY( 'F', N, N, VL, LDVL, VR, LDVR )
          END IF
*
      ELSE IF( WANTVR ) THEN
*
*          Want right eigenvectors
*          Copy Householder vectors to VR
*
          SIDE = 'R'
          CALL ZLACPY( 'L', N, N, A, LDA, VR, LDVR )
*
*          Generate unitary matrix in VR
*          (CWorkspace: need 2*N-1, prefer N+(N-1)*NB)
*          (RWorkspace: none)

```



```

*
*      CALL ZUNGHR( N, ILO, IHI, VR, LDVR, WORK( ITAU ), WORK( IWRK ),
$          LWORK-IWRK+1, IERR )
*
*      Perform QR iteration, accumulating Schur vectors in VR
*      (CWorkspace: need 1, prefer HSWORK (see comments) )
*      (RWorkspace: none)
*
*      IWRK = ITAU
*      CALL ZHSEQR( 'S', 'V', N, ILO, IHI, A, LDA, W, VR, LDVR,
$          WORK( IWRK ), LWORK-IWRK+1, INFO )
*
*      ELSE
*
*      Compute eigenvalues only
*      (CWorkspace: need 1, prefer HSWORK (see comments) )
*      (RWorkspace: none)
*
*      IWRK = ITAU
*      CALL ZHSEQR( 'E', 'N', N, ILO, IHI, A, LDA, W, VR, LDVR,
$          WORK( IWRK ), LWORK-IWRK+1, INFO )
*      END IF
*
*      If INFO > 0 from ZHSEQR, then quit
*
*      IF( INFO.GT.0 )
$      GO TO 50
*
*      IF( WANTVL .OR. WANTVR ) THEN
*
*      Compute left and/or right eigenvectors
*      (CWorkspace: need 2*N)
*      (RWorkspace: need 2*N)
*
*      IRWORK = IBAL + N
*      CALL ZTREVC( SIDE, 'B', SELECT, N, A, LDA, VL, LDVL, VR, LDVR,
$          N, NOUT, WORK( IWRK ), RWORK( IRWORK ), IERR )
*      END IF
*
*      IF( WANTVL ) THEN
*
*      Undo balancing of left eigenvectors
*      (CWorkspace: none)
*      (RWorkspace: need N)
*
*      CALL ZGEBAK( 'B', 'L', N, ILO, IHI, RWORK( IBAL ), N, VL, LDVL,
$          IERR )
*
*      Normalize left eigenvectors and make largest component real
*

```

```

DO 20 I = 1, N
    SCL = ONE / DZNRM2( N, VL( 1, I ), 1 )
    CALL ZDSCAL( N, SCL, VL( 1, I ), 1 )
    DO 10 K = 1, N
        RWORK( IRWORK+K-1 ) = DBLE( VL( K, I ) )**2 +
$                               DIMAG( VL( K, I ) )**2
10    CONTINUE
        K = IDAMAX( N, RWORK( IRWORK ), 1 )
        TMP = DCONJG( VL( K, I ) ) / SQRT( RWORK( IRWORK+K-1 ) )
        CALL ZSCAL( N, TMP, VL( 1, I ), 1 )
        VL( K, I ) = DCMPLX( DBLE( VL( K, I ) ), ZERO )
20    CONTINUE
    END IF

*
    IF( WANTVR ) THEN
*
*       Undo balancing of right eigenvectors
*       (CWorkspace: none)
*       (RWorkspace: need N)
*
        CALL ZGEBAK( 'B', 'R', N, ILO, IHI, RWORK( IBAL ), N, VR, LDVR,
$                   IERR )
*
*       Normalize right eigenvectors and make largest component real
*
        DO 40 I = 1, N
            SCL = ONE / DZNRM2( N, VR( 1, I ), 1 )
            CALL ZDSCAL( N, SCL, VR( 1, I ), 1 )
            DO 30 K = 1, N
                RWORK( IRWORK+K-1 ) = DBLE( VR( K, I ) )**2 +
$                               DIMAG( VR( K, I ) )**2
30        CONTINUE
            K = IDAMAX( N, RWORK( IRWORK ), 1 )
            TMP = DCONJG( VR( K, I ) ) / SQRT( RWORK( IRWORK+K-1 ) )
            CALL ZSCAL( N, TMP, VR( 1, I ), 1 )
            VR( K, I ) = DCMPLX( DBLE( VR( K, I ) ), ZERO )
40        CONTINUE
        END IF

*
*       Undo scaling if necessary
*
50    CONTINUE
    IF( SCALEA ) THEN
        CALL ZLASCL( 'G', 0, 0, CSCALE, ANRM, N-INFO, 1, W( INFO+1 ),
$                   MAX( N-INFO, 1 ), IERR )
        IF( INFO.GT.0 ) THEN
            CALL ZLASCL( 'G', 0, 0, CSCALE, ANRM, ILO-1, 1, W, N, IERR )
        END IF
    END IF

*

```

```

      WORK( 1 ) = MAXWRK
      RETURN
*
*      End of ZGEEV
*
      END

```

— LAPACK zggev —

```

(let* ((zero 0.0d0) (one 1.0d0))
  (declare (type (double-float 0.0d0 0.0d0) zero)
    (type (double-float 1.0d0 1.0d0) one) (ignorable zero one))
  (defun zggev (jobvl jobvr n a lda w vl ldvl vr ldvr work lwork rwork info)
    (declare (type (simple-array character (*)) jobvr jobvl)
      (type (f2cl-lib:integer4) info lwork ldvr ldvl lda n)
      (type (array f2cl-lib:complex16 (*)) work vr vl w a)
      (type (array double-float (*)) rwork))
    (f2cl-lib:with-multi-array-data
      ((rwork double-float rwork-%data%
        rwork-%offset%)
        (a f2cl-lib:complex16 a-%data% a-%offset%)
        (w f2cl-lib:complex16 w-%data% w-%offset%)
        (vl f2cl-lib:complex16 vl-%data% vl-%offset%)
        (vr f2cl-lib:complex16 vr-%data% vr-%offset%)
        (work f2cl-lib:complex16 work-%data% work-%offset%)
        (jobvl character jobvl-%data% jobvl-%offset%)
        (jobvr character jobvr-%data% jobvr-%offset%))
      (prog
        ((dum (make-array 1 :element-type 'double-float))
         (select (make-array 1 :element-type 't))
         (tmp #C(0.0d0 0.0d0)) (anrm 0.0d0)
         (bignum 0.0d0) (cscale 0.0d0) (eps 0.0d0) (scl 0.0d0) (smlnum 0.0d0)
         (hswork 0) (i 0) (ibal 0) (ierr 0) (ihi 0) (ilo 0) (irwork 0) (itau 0)
         (iwrk 0) (k 0) (maxwrk 0) (minwrk 0) (nout 0)
         (side (make-array '(1) :element-type 'character
                           :initial-element #\space))
         (lquery nil) (scalea nil) (wantvl nil) (wantvr nil))
        (declare (type (array double-float (1)) dum)
          (type (array f2cl-lib:logical (1)) select)
          (type (f2cl-lib:complex16) tmp)
          (type (double-float) smlnum scl eps cscale bignum anrm)
          (type (f2cl-lib:integer4) nout minwrk maxwrk k iwrk itau irwork
            ilo ihi ierr
            ibal i hswork)
          (type (simple-array character (1)) side)
          (type f2cl-lib:logical wantvr wantvl scalea lquery))

```

```

(setf info 0) (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical))
(setf wantvl
  (multiple-value-bind (ret-val var-0 var-1) (lsame jobvl "V")
    (declare (ignore var-1)) (when var-0 (setf jobvl var-0)) ret-val))
(setf wantvr
  (multiple-value-bind (ret-val var-0 var-1) (lsame jobvr "V")
    (declare (ignore var-1)) (when var-0 (setf jobvr var-0)) ret-val))
(cond
  ((and (not wantvl)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame jobvl "N")
        (declare (ignore var-1))
        (when var-0 (setf jobvl var-0)) ret-val)))
    (setf info -1))
  ((and (not wantvr)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame jobvr "N")
        (declare (ignore var-1))
        (when var-0 (setf jobvr var-0)) ret-val)))
    (setf info -2))
  ((< n 0) (setf info -3))
  ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
    (setf info -5))
  ((or (< ldvl 1) (and wantvl (< ldvl n))) (setf info -8))
  ((or (< ldvr 1) (and wantvr (< ldvr n))) (setf info -10)))
(cond
  ((= info 0)
    (cond ((= n 0) (setf minwrk 1) (setf maxwrk 1))
    (t
      (setf maxwrk
        (f2cl-lib:int-add n
          (f2cl-lib:int-mul n
            (multiple-value-bind
              (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
              (ilaenv 1 "ZGEHRD" " " n 1 n 0)
              (declare (ignore var-0 var-1 var-2 var-4 var-6))
              (when var-3 (setf n var-3))
              (when var-5 (setf n var-5)) ret-val))))
        (setf minwrk (f2cl-lib:int-mul 2 n))
      (cond
        (wantvl
          (setf maxwrk
            (max (the f2cl-lib:integer4 maxwrk)
              (the f2cl-lib:integer4
                (f2cl-lib:int-add n
                  (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                    (multiple-value-bind
                      (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                      (ilaenv 1 "ZUNGHR" " " n 1 n -1)
                      (declare (ignore var-0 var-1 var-2 var-4 var-6))

```

```

        (when var-3 (setf n var-3)) (when var-5 (setf n var-5))
        ret-val))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11 var-12)
  (zhseqr "S" "V" n 1 n a lda w vl ldvl work -1 info)
  (declare (ignore var-0 var-1 var-3 var-5 var-7 var-8 var-10
                  var-11))
  (when var-2 (setf n var-2)) (when var-4 (setf n var-4))
  (when var-6 (setf lda var-6)) (when var-9 (setf ldvl var-9))
  (when var-12 (setf info var-12))))
(wantvr
  (setf maxwrk
    (max (the f2cl-lib:integer4 maxwrk)
         (the f2cl-lib:integer4
           (f2cl-lib:int-add n
            (f2cl-lib:int-mul (f2cl-lib:int-sub n 1)
                              (multiple-value-bind
                                (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                                (ilaenv 1 "ZUNGHR" " " n 1 n -1)
                                (declare (ignore var-0 var-1 var-2 var-4 var-6))
                                (when var-3 (setf n var-3)) (when var-5 (setf n var-5))
                                ret-val)))))))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
     var-10 var-11 var-12)
    (zhseqr "S" "V" n 1 n a lda w vr ldvr work -1 info)
    (declare (ignore var-0 var-1 var-3 var-5 var-7 var-8 var-10
                    var-11))
    (when var-2 (setf n var-2)) (when var-4 (setf n var-4))
    (when var-6 (setf lda var-6)) (when var-9 (setf ldvr var-9))
    (when var-12 (setf info var-12))))
  (t
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
       var-10 var-11 var-12)
      (zhseqr "E" "N" n 1 n a lda w vr ldvr work -1 info)
      (declare (ignore var-0 var-1 var-3 var-5 var-7 var-8 var-10
                      var-11))
      (when var-2 (setf n var-2)) (when var-4 (setf n var-4))
      (when var-6 (setf lda var-6)) (when var-9 (setf ldvr var-9))
      (when var-12 (setf info var-12))))
    (setf hswork
      (f2cl-lib:int (f2cl-lib:fref work-%data% (1) ((1 *))
                            work-%offset%)))
    (setf maxwrk
      (max (the f2cl-lib:integer4 maxwrk) (the f2cl-lib:integer4 hswork))

```

```

        (the f2cl-lib:integer4 minwrk))))))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce maxwrk 'f2cl-lib:complex16))
    (cond ((and (< lwork minwrk) (not lquery)) (setf info -12))))))
(cond ((/= info 0)
  (xerbla "ZGEEV " (f2cl-lib:int-sub info))
  (go end_label))
  (lquery (go end_label)))
(if (= n 0) (go end_label))
(setf eps (dlamch "P")) (setf smlnum (dlamch "S"))
(setf bignum (/ one smlnum))
(multiple-value-bind (var-0 var-1)
  (dlabad smlnum bignum) (declare (ignore)))
  (when var-0 (setf smlnum var-0)) (when var-1 (setf bignum var-1))
(setf smlnum (/ (f2cl-lib:fsqrt smlnum) eps))
(setf bignum (/ one smlnum))
(setf anrm
  (multiple-value-bind (ret-val var-0 var-1 var-2 var-3 var-4 var-5)
    (zlange "M" n n a lda dum) (declare (ignore var-0 var-3 var-5))
    (when var-1 (setf n var-1)) (when var-2 (setf n var-2))
    (when var-4 (setf lda var-4)) ret-val))
(setf scalea f2cl-lib:%false%)
(cond
  ((and (> anrm zero) (< anrm smlnum)) (setf scalea f2cl-lib:%true%)
   (setf cscale smlnum))
  ((> anrm bignum) (setf scalea f2cl-lib:%true%) (setf cscale bignum)))
(if scalea
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (zlascl "G" 0 0 anrm cscale n n a lda ierr)
    (declare (ignore var-0 var-1 var-2 var-7))
    (when var-3 (setf anrm var-3))
    (when var-4 (setf cscale var-4)) (when var-5 (setf n var-5))
    (when var-6 (setf n var-6)) (when var-8 (setf lda var-8))
    (when var-9 (setf ierr var-9))))
  (setf ibal 1)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
    (zgebal "B" n a lda ilo ihi
      (f2cl-lib:array-slice rwork-%data% double-float (ibal) ((1 *))
        rwork-%offset%)
      ierr)
    (declare (ignore var-0 var-1 var-2 var-6))
    (setf lda var-3) (setf ilo var-4)
    (setf ihi var-5) (setf ierr var-7))
    (setf itau 1) (setf iwrk (f2cl-lib:int-add itau n))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (zgehrd n ilo ihi a lda
        (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (itau) ((1 *))
          work-%offset%))

```

```

(f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
 work-%offset%)
(f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
(declare (ignore var-3 var-5 var-6 var-7)) (when var-0 (setf n var-0))
(when var-1 (setf ilo var-1)) (when var-2 (setf ihi var-2))
(when var-4 (setf lda var-4)) (when var-8 (setf ierr var-8)))
(cond
 (wantvl (f2cl-lib:f2cl-set-string side "L" (string 1))
 (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (zlacpy "L" n n a lda vl ldvl) (declare (ignore var-0 var-3 var-5))
  (when var-1 (setf n var-1)) (when var-2 (setf n var-2))
  (when var-4 (setf lda var-4)) (when var-6 (setf ldvl var-6)))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zunghr n ilo ihi vl ldvl
   (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (itau) ((1 *))
    work-%offset%)
   (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
    work-%offset%)
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
  (declare (ignore var-3 var-5 var-6 var-7))
  (when var-0 (setf n var-0))
  (when var-1 (setf ilo var-1)) (when var-2 (setf ihi var-2))
  (when var-4 (setf ldvl var-4)) (when var-8 (setf ierr var-8)))
  (setf iwrk itau)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11
   var-12)
  (zhseqr "S" "V" n ilo ihi a lda w vl ldvl
   (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
    work-%offset%)
   (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
  (declare (ignore var-0 var-1 var-5 var-7 var-8 var-10 var-11))
  (when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
  (when var-4 (setf ihi var-4)) (when var-6 (setf lda var-6))
  (when var-9 (setf ldvl var-9)) (when var-12 (setf info var-12)))
 (cond
  (wantvr (f2cl-lib:f2cl-set-string side "B" (string 1))
   (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (zlacpy "F" n n vl ldvl vr ldvr)
    (declare (ignore var-0 var-3 var-5))
    (when var-1 (setf n var-1)) (when var-2 (setf n var-2))
    (when var-4 (setf ldvl var-4)) (when var-6 (setf ldvr var-6))))))
 (wantvr (f2cl-lib:f2cl-set-string side "R" (string 1))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
   (zlacpy "L" n n a lda vr ldvr) (declare (ignore var-0 var-3 var-5))
   (when var-1 (setf n var-1)) (when var-2 (setf n var-2))
   (when var-4 (setf lda var-4)) (when var-6 (setf ldvr var-6)))
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6

```

```

var-7 var-8)
(zunghr n ilo ihi vr ldvr
  (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (itau) ((1 *))
    work-%offset%)
  (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
    work-%offset%)
  (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) ierr)
(declare (ignore var-3 var-5 var-6 var-7))
(when var-0 (setf n var-0))
(when var-1 (setf ilo var-1)) (when var-2 (setf ihi var-2))
(when var-4 (setf ldvr var-4)) (when var-8 (setf ierr var-8)))
(setf iwrk itau)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
   var-10 var-11
   var-12)
  (zhseqr "S" "V" n ilo ihi a lda w vr ldvr
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
      work-%offset%)
    (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
  (declare (ignore var-0 var-1 var-5 var-7 var-8 var-10 var-11))
  (when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
  (when var-4 (setf ihi var-4)) (when var-6 (setf lda var-6))
  (when var-9 (setf ldvr var-9)) (when var-12 (setf info var-12))))
(t (setf iwrk itau)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
     var-10 var-11
     var-12)
    (zhseqr "E" "N" n ilo ihi a lda w vr ldvr
      (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
        work-%offset%)
      (f2cl-lib:int-add (f2cl-lib:int-sub lwork iwrk) 1) info)
    (declare (ignore var-0 var-1 var-5 var-7 var-8 var-10 var-11))
    (when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
    (when var-4 (setf ihi var-4)) (when var-6 (setf lda var-6))
    (when var-9 (setf ldvr var-9)) (when var-12 (setf info var-12))))))
(if (> info 0) (go label50))
(cond
  ((or wantvl wantvr) (setf irwork (f2cl-lib:int-add ibal n))
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
      var-9 var-10 var-11
      var-12 var-13 var-14)
     (ztrevc side "B" select n a lda vl ldvl vr ldvr n nout
       (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (iwrk) ((1 *))
         work-%offset%)
       (f2cl-lib:array-slice rwork-%data% double-float (irwork) ((1 *))
         rwork-%offset%)
       ierr)

```



```

(declare (ignore var-1 var-2 var-4 var-6 var-8 var-12 var-13))
(when var-0 (setf side var-0)) (when var-3 (setf n var-3))
(when var-5 (setf lda var-5)) (when var-7 (setf ldvl var-7))
(when var-9 (setf ldvr var-9)) (when var-10 (setf n var-10))
(when var-11 (setf nout var-11)) (when var-14 (setf ierr var-14))))
(cond
  (wantvl
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (zgebak "B" "L" n ilo ihi
        (f2cl-lib:array-slice rwork-%data% double-float (ibal) ((1 *))
          rwork-%offset%)
          n vl ldvl ierr)
      (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
      (setf n var-6)
      (setf ldvl var-8) (setf ierr var-9))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i n) nil)
      (tagbody
        (setf scl
          (/ one
            (multiple-value-bind (ret-val var-0 var-1 var-2)
              (dznrm2 n
                (f2cl-lib:array-slice vl-%data% f2cl-lib:complex16 (1 i)
                  ((1 ldvl) (1 *)) vl-%offset%)
                  1)
              (declare (ignore var-1 var-2))
              (when var-0 (setf n var-0)) ret-val))))
          (multiple-value-bind (var-0 var-1 var-2 var-3)
            (zdscal n scl
              (f2cl-lib:array-slice vl-%data% f2cl-lib:complex16
                (1 i) ((1 ldvl) (1 *))
                vl-%offset%)
              1)
            (declare (ignore var-2 var-3))
            (when var-0 (setf n var-0))
            (when var-1 (setf scl var-1)))
            (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
              (> k n) nil)
            (tagbody
              (setf
                (f2cl-lib:fref rwork-%data%
                  ((f2cl-lib:int-sub (f2cl-lib:int-add irwork k) 1))
                  ((1 *))
                  rwork-%offset%)
                (+
                  (expt
                    (f2cl-lib:dbler
                      (f2cl-lib:fref vl-%data% (k i) ((1 ldvl) (1 *))
                        vl-%offset%))

```

```

2)
(expt
  (f2cl-lib:dimag
    (f2cl-lib:fref vl-%data% (k i) ((1 ldvl) (1 *))
    vl-%offset%))
  2)))
label10))
(setf k
  (multiple-value-bind (ret-val var-0 var-1 var-2)
    (idamax n
      (f2cl-lib:array-slice rwork-%data% double-float
        (irwork) ((1 *))
        rwork-%offset%))
    1)
  (declare (ignore var-1 var-2))
  (when var-0 (setf n var-0)) ret-val))
(setf tmp
  (coerce
    (/
      (f2cl-lib:dconjg
        (f2cl-lib:fref vl-%data% (k i) ((1 ldvl) (1 *))
        vl-%offset%))
      (f2cl-lib:fsqrt
        (f2cl-lib:fref rwork-%data%
          ((f2cl-lib:int-sub (f2cl-lib:int-add irwork k) 1))
          ((1 *))
          rwork-%offset%))))
    'f2cl-lib:complex16))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal n tmp
    (f2cl-lib:array-slice vl-%data% f2cl-lib:complex16
      (1 i) ((1 ldvl) (1 *))
      vl-%offset%))
  1)
  (declare (ignore var-2 var-3))
  (when var-0 (setf n var-0))
  (when var-1 (setf tmp var-1)))
(setf (f2cl-lib:fref vl-%data% (k i) ((1 ldvl) (1 *))
  vl-%offset%))
(f2cl-lib:dcmplx
  (f2cl-lib:dbler
    (f2cl-lib:fref vl-%data% (k i) ((1 ldvl) (1 *))
    vl-%offset%))
  zero))
label20))))
(cond
  (wantvr
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (zgebak "B" "R" n ilo ihi

```

```

(f2cl-lib:array-slice rwork-%data% double-float (ibal) ((1 *))
 rwork-%offset%)
n vr ldvr ierr)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7))
(setf n var-6)
(setf ldvr var-8) (setf ierr var-9))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
 (> i n) nil)
  (tagbody
    (setf scl
      (/ one
        (multiple-value-bind (ret-val var-0 var-1 var-2)
          (dznrm2 n
            (f2cl-lib:array-slice vr-%data% f2cl-lib:complex16
              (1 i)
              ((1 ldvr) (1 *)) vr-%offset%)
            1)
          (declare (ignore var-1 var-2))
          (when var-0 (setf n var-0)) ret-val)))
      (multiple-value-bind (var-0 var-1 var-2 var-3)
        (zdscal n scl
          (f2cl-lib:array-slice vr-%data% f2cl-lib:complex16
            (1 i) ((1 ldvr) (1 *))
            vr-%offset%)
          1)
        (declare (ignore var-2 var-3))
        (when var-0 (setf n var-0))
        (when var-1 (setf scl var-1))))
      (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
        (> k n) nil)
      (tagbody
        (setf
          (f2cl-lib:fref rwork-%data%
            ((f2cl-lib:int-sub (f2cl-lib:int-add irwork k) 1))
            ((1 *))
            rwork-%offset%)
          (+
            (expt
              (f2cl-lib:dbler
                (f2cl-lib:fref vr-%data% (k i) ((1 ldvr) (1 *))
                  vr-%offset%))
              2)
            (expt
              (f2cl-lib:dimag
                (f2cl-lib:fref vr-%data% (k i) ((1 ldvr) (1 *))
                  vr-%offset%))
              2))))
          label30))
      (setf k
        (multiple-value-bind (ret-val var-0 var-1 var-2)

```

```

(idamax n
  (f2cl-lib:array-slice rwork-%data% double-float
    (irwork) ((1 *))
    rwork-%offset%)
  1)
(declare (ignore var-1 var-2))
(when var-0 (setf n var-0)) ret-val))
(setf tmp
  (coerce
    (/
      (f2cl-lib:dconjg
        (f2cl-lib:fref vr-%data% (k i) ((1 ldvr) (1 *))
        vr-%offset%))
      (f2cl-lib:fsqrt
        (f2cl-lib:fref rwork-%data%
          ((f2cl-lib:int-sub (f2cl-lib:int-add irwork k) 1))
          ((1 *))
          rwork-%offset%))))
    'f2cl-lib:complex16))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal n tmp
    (f2cl-lib:array-slice vr-%data% f2cl-lib:complex16
      (1 i) ((1 ldvr) (1 *))
      vr-%offset%)
    1)
  (declare (ignore var-2 var-3)) (when var-0 (setf n var-0))
  (when var-1 (setf tmp var-1)))
(setf (f2cl-lib:fref vr-%data% (k i) ((1 ldvr) (1 *))
  vr-%offset%)
  (f2cl-lib:dcmplx
    (f2cl-lib:dbble
      (f2cl-lib:fref vr-%data% (k i) ((1 ldvr) (1 *))
        vr-%offset%))
    zero))
label40)))

label150
(cond
  (scalea
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
      (zlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub n info) 1
        (f2cl-lib:array-slice w-%data% f2cl-lib:complex16
          ((+ info 1)) ((1 *))
          w-%offset%)
        (max (the f2cl-lib:integer4 (f2cl-lib:int-sub n info))
          (the f2cl-lib:integer4 1))
        ierr)
      (declare (ignore var-0 var-1 var-2 var-5 var-6 var-7 var-8))
      (when var-3 (setf cscale var-3)) (when var-4 (setf anrm var-4))
      (when var-9 (setf ierr var-9)))

```

```

(cond
  (> info 0)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9)
    (zlascl "G" 0 0 cscale anrm (f2cl-lib:int-sub ilo 1) 1 w n ierr)
    (declare (ignore var-0 var-1 var-2 var-5 var-6 var-7))
    (when var-3 (setf cscale var-3)) (when var-4 (setf anrm var-4))
    (when var-8 (setf n var-8)) (when var-9 (setf ierr var-9))))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce maxwrk 'f2cl-lib:complex16))
(go end_label) end_label
(return
  (values jobvl jobvr n nil lda nil nil ldvl nil
          ldvr nil nil nil info)))
)))

```

zgehd2 LAPACK

— zgehd2.input —

```

)set break resume
)sys rm -f zgehd2.output
)spool zgehd2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zgehd2.help —

```

=====
zgehd2 examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```
SUBROUTINE ZGEHD2( N, ILO, IHI, A, LDA, TAU, WORK, INFO )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHI, ILO, INFO, LDA, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
```

```
..
```

Purpose:

=====

ZGEHD2 reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation: $Q^*H * A * Q = H$.

Arguments:

=====

[in] N

N is INTEGER

The order of the matrix A. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.
 $1 \leq \text{ILO} \leq \text{IHI} \leq \max(1, N)$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)

On entry, the n by n general matrix to be reduced.

On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary

reflectors. See Further Details.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. $LDA \geq \max(1, N)$.

[out] TAU

TAU is COMPLEX*16 array, dimension (N-1)
The scalar factors of the elementary reflectors (see Further Details).

[out] WORK

WORK is COMPLEX*16 array, dimension (N)

[out] INFO

INFO is INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value.

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Further Details:

=====

The matrix Q is represented as a product of (ihi-ilo) elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v^* H$$

where tau is a complex scalar, and v is a complex vector with $v(1:i) = 0$, $v(i+1) = 1$ and $v(ihi+1:n) = 0$; $v(i+2:ihi)$ is stored on exit in $A(i+2:ihi, i)$, and tau in $TAU(i)$.

The contents of A are illustrated by the following example, with

$n = 7$, $ilo = 2$ and $ihi = 6$:

| on entry, | on exit, |
|-------------------|--------------------|
| (a a a a a a a) | (a a h h h h a) |
| (a a a a a a a) | (a h h h h a) |
| (a a a a a a a) | (h h h h h h) |
| (a a a a a a a) | (v2 h h h h h) |
| (a a a a a a a) | (v2 v3 h h h h) |
| (a a a a a a a) | (v2 v3 v4 h h h) |
| (a) | (a) |

where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and v_i denotes an element of the vector defining $H(i)$.

— zgehd2.f —

```

* =====
*      SUBROUTINE ZGEHD2( N, ILO, IHI, A, LDA, TAU, WORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, ILO, INFO, LDA, N
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
*      ..
*
*      =====
*
*      .. Parameters ..
*      COMPLEX*16       ONE
*      PARAMETER        ( ONE = ( 1.0D+0, 0.0D+0 ) )
*      ..
*      .. Local Scalars ..
*      INTEGER          I
*      COMPLEX*16       ALPHA
*      ..
*      .. External Subroutines ..
*      EXTERNAL         XERBLA, ZLARF, ZLARFG
*      ..

```



```

*      .. Intrinsic Functions ..
      INTRINSIC          DCONJG, MAX, MIN
*
*      ..
*      .. Executable Statements ..
*
*      Test the input parameters
*
      INFO = 0
      IF( N.LT.0 ) THEN
        INFO = -1
      ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
        INFO = -2
      ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -3
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZGEHD2', -INFO )
        RETURN
      END IF
*
      DO 10 I = ILO, IHI - 1
*
*        Compute elementary reflector H(i) to annihilate A(i+2:ihi,i)
*
        ALPHA = A( I+1, I )
        CALL ZLARFG( IHI-I, ALPHA, A( MIN( I+2, N ), I ), 1, TAU( I ) )
        A( I+1, I ) = ONE
*
*        Apply H(i) to A(1:ihi,i+1:ihi) from the right
*
        CALL ZLARF( 'Right', IHI, IHI-I, A( I+1, I ), 1, TAU( I ),
$              A( 1, I+1 ), LDA, WORK )
*
*        Apply H(i)**H to A(i+1:ihi,i+1:n) from the left
*
        CALL ZLARF( 'Left', IHI-I, N-I, A( I+1, I ), 1,
$              DCONJG( TAU( I ) ), A( I+1, I+1 ), LDA, WORK )
*
        A( I+1, I ) = ALPHA
10 CONTINUE
*
      RETURN
*
*      End of ZGEHD2
*
      END

```

— LAPACK zgehd2 —

```

(let* ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (ignorable one))
  (defun zgehd2 (n ilo ihi a lda tau work info)
    (declare (type (f2cl-lib:integer4) info lda ihi ilo n)
      (type (array f2cl-lib:complex16 (*)) work tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (prog
        ((alpha #C(0.0d0 0.0d0)) (i 0) (dconjg$ 0.0))
        (declare (type (f2cl-lib:complex16) alpha) (type (f2cl-lib:integer4) i)
          (type (single-float) dconjg$))
        (setf info 0)
        (cond ((< n 0) (setf info -1))
          ((or (< ilo 1)
            (> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))
            (setf info -2))
          ((or (< ihi (min (the f2cl-lib:integer4 ilo)
              (the f2cl-lib:integer4 n)))
            (> ihi n))
            (setf info -3))
          ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
            (setf info -5)))
        (cond ((/= info 0)
          (xerbla "ZGEHD2" (f2cl-lib:int-sub info))
          (go end_label)))
        (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i 1))
          ((> i
            (f2cl-lib:int-add ihi (f2cl-lib:int-sub 1)))
            nil)
          (tagbody
            (setf alpha
              (f2cl-lib:fref a-%data% ((f2cl-lib:int-add i 1) i)
                ((1 lda) (1 *))
                a-%offset%))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (zlarfg (f2cl-lib:int-sub ihi i) alpha
                (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
                  ((min (f2cl-lib:int-add i 2) n) i) ((1 lda) (1 *))
                  a-%offset%))
                1
              (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
                (i) ((1 *))
                tau-%offset%))

```

```

(declare (ignore var-0 var-2 var-3 var-4))
(when var-1 (setf alpha var-1)))
(setf
  (f2cl-lib:fref a-%data% ((f2cl-lib:int-add i 1) i)
    ((1 lda) (1 *))
    a-%offset%)
  one)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8)
  (zlarf "Right" ihi (f2cl-lib:int-sub ihi i)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ i 1) i)
      ((1 lda) (1 *)) a-%offset%)
    1
    (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
      (i) ((1 *))
      tau-%offset%)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      (1 (f2cl-lib:int-add i 1)) ((1 lda) (1 *)) a-%offset%)
    lda work)
  (declare (ignore var-0 var-2 var-3 var-4 var-5 var-6 var-8))
  (when var-1 (setf ihi var-1)) (when var-7 (setf lda var-7)))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8)
  (zlarf "Left" (f2cl-lib:int-sub ihi i)
    (f2cl-lib:int-sub n i)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ i 1) i)
      ((1 lda) (1 *)) a-%offset%)
    1 (f2cl-lib:dconjg (f2cl-lib:fref tau-%data% (i) ((1 *))
      tau-%offset%))
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ i 1) (f2cl-lib:int-add i 1)) ((1 lda) (1 *))
      a-%offset%)
    lda work)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-5
    var-6 var-8))
  (when var-7 (setf lda var-7)))
(setf
  (f2cl-lib:fref a-%data% ((f2cl-lib:int-add i 1) i)
    ((1 lda) (1 *))
    a-%offset%)
  alpha)
label10))
(go end_label)
end_label
(return (values nil nil ihi nil lda nil nil info)))
)))

```

zgehrd LAPACK

— zgehrd.input —

```
)set break resume
)sys rm -f zgehrd.output
)spool zgehrd.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zgehrd.help —

```
=====
zgehrd examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
SUBROUTINE ZGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHI, ILO, INFO, LDA, LWORK, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
```

```
..
```

Purpose:

=====

ZGEHRD reduces a complex general matrix A to upper Hessenberg form H by
 an unitary similarity transformation: $Q^* H * A * Q = H$.

Arguments:

=====

[in] N

N is INTEGER

The order of the matrix A. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.

$1 \leq \text{ILO} \leq \text{IHI} \leq N$, if $N > 0$; ILO=1 and IHI=0, if $N=0$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)

On entry, the N-by-N general matrix to be reduced.

On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

[in] LDA

LDA is INTEGER

The leading dimension of the array A. $\text{LDA} \geq \max(1, N)$.

[out] TAU

TAU is COMPLEX*16 array, dimension (N-1)

The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

[out] WORK

WORK is COMPLEX*16 array, dimension (LWORK)

On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER

The length of the array WORK. $LWORK \geq \max(1, N)$.

For optimum performance $LWORK \geq N \cdot NB$, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

[out] INFO

INFO is INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:

=====

The matrix Q is represented as a product of (ihi-ilo) elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v^* H$$

where tau is a complex scalar, and v is a complex vector with $v(1:i) = 0$, $v(i+1) = 1$ and $v(ihi+1:n) = 0$; $v(i+2:ihi)$ is stored on exit in $A(i+2:ihi, i)$, and tau in $TAU(i)$.

The contents of A are illustrated by the following example, with $n = 7$, $ilo = 2$ and $ihi = 6$:

on entry,

on exit,

| | |
|-------------------------------|-------------------------------|
| (a a a a a a a) | (a a h h h h a) |
| (a a a a a a) | (a h h h h a) |

```

(   a   a   a   a   a   a   )   (   h   h   h   h   h   h   )
(   a   a   a   a   a   a   )   (   v2  h   h   h   h   h   )
(   a   a   a   a   a   a   )   (   v2  v3  h   h   h   h   )
(   a   a   a   a   a   a   )   (   v2  v3  v4  h   h   h   )
(                               )   (                               a   )

```

where a denotes an element of the original matrix A, h denotes a modified element of the upper Hessenberg matrix H, and vi denotes an element of the vector defining H(i).

This file is a slight modification of LAPACK-3.0's DGEHRD subroutine incorporating improvements proposed by Quintana-Orti and Van de Geijn (2006). (See DLAHR2.)

— zgehrd.f —

```

* =====
*      SUBROUTINE ZGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,      --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, ILO, INFO, LDA, LWORK, N
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
*      ..
*
*      =====
*
*      .. Parameters ..
*      INTEGER          NBMAX, LDT
*      PARAMETER        ( NBMAX = 64, LDT = NBMAX+1 )
*      COMPLEX*16       ZERO, ONE
*      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ),
*      $                ONE = ( 1.0D+0, 0.0D+0 ) )
*      ..
*      .. Local Scalars ..
*      LOGICAL          LQUERY
*      INTEGER          I, IB, IINFO, IWS, J, LDWORK, LWKOPT, NB,
*      $                NBMIN, NH, NX
*      COMPLEX*16       EI
*      ..

```

```

*    .. Local Arrays ..
      COMPLEX*16      T( LDT, NBMAX )
*
*    ..
*    .. External Subroutines ..
      EXTERNAL        ZAXPY, ZGEHD2, ZGEMM, ZLAHR2, ZLARFB, ZTRMM,
$                    XERBLA
*
*    ..
*    .. Intrinsic Functions ..
      INTRINSIC        MAX, MIN
*
*    ..
*    .. External Functions ..
      INTEGER          ILAENV
      EXTERNAL          ILAENV
*
*    ..
*    .. Executable Statements ..
*
*    Test the input parameters
*
      INFO = 0
      NB = MIN( NBMAX, ILAENV( 1, 'ZGEHRD', ' ', N, ILO, IHI, -1 ) )
      LWKOPT = N*NB
      WORK( 1 ) = LWKOPT
      LQUERY = ( LWORK.EQ.-1 )
      IF( N.LT.0 ) THEN
        INFO = -1
      ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
        INFO = -2
      ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -3
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -5
      ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
        INFO = -8
      END IF
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZGEHRD', -INFO )
        RETURN
      ELSE IF( LQUERY ) THEN
        RETURN
      END IF
*
*    Set elements 1:ILO-1 and IHI:N-1 of TAU to zero
*
      DO 10 I = 1, ILO - 1
        TAU( I ) = ZERO
10  CONTINUE
      DO 20 I = MAX( 1, IHI ), N - 1
        TAU( I ) = ZERO
20  CONTINUE
*

```



```

*      Quick return if possible
*
      NH = IHI - ILO + 1
      IF( NH.LE.1 ) THEN
        WORK( 1 ) = 1
        RETURN
      END IF
*
*      Determine the block size
*
      NB = MIN( NBMAX, ILAENV( 1, 'ZGEHRD', ' ', N, ILO, IHI, -1 ) )
      NBMIN = 2
      IWS = 1
      IF( NB.GT.1 .AND. NB.LT.NH ) THEN
*
*      Determine when to cross over from blocked to unblocked code
*      (last block is always handled by unblocked code)
*
      NX = MAX( NB, ILAENV( 3, 'ZGEHRD', ' ', N, ILO, IHI, -1 ) )
      IF( NX.LT.NH ) THEN
*
*      Determine if workspace is large enough for blocked code
*
      IWS = N*NB
      IF( LWORK.LT.IWS ) THEN
*
*      Not enough workspace to use optimal NB:  determine the
*      minimum value of NB, and reduce NB or force use of
*      unblocked code
*
      NBMIN = MAX( 2, ILAENV( 2, 'ZGEHRD', ' ', N, ILO, IHI,
$      -1 ) )
      IF( LWORK.GE.N*NBMIN ) THEN
        NB = LWORK / N
      ELSE
        NB = 1
      END IF
      END IF
      END IF
      LDWORK = N
*
      IF( NB.LT.NBMIN .OR. NB.GE.NH ) THEN
*
*      Use unblocked code below
*
      I = ILO
*
      ELSE
*

```

```

*      Use blocked code
*
      DO 40 I = ILO, IHI - 1 - NX, NB
          IB = MIN( NB, IHI-I )
*
*      Reduce columns i:i+ib-1 to Hessenberg form, returning the
*      matrices V and T of the block reflector  $H = I - V^*T^*V$ 
*      which performs the reduction, and also the matrix  $Y = A^*V^*T$ 
*
          CALL ZLAHR2( IHI, I, IB, A( 1, I ), LDA, TAU( I ), T, LDT,
$              WORK, LDWORK )
*
*      Apply the block reflector H to A(1:ihi,i+ib:ihi) from the
*      right, computing  $A := A - Y * V^*H$ . V(i+ib,ib-1) must be set
*      to 1
*
          EI = A( I+IB, I+IB-1 )
          A( I+IB, I+IB-1 ) = ONE
          CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$              IHI, IHI-I-IB+1,
$              IB, -ONE, WORK, LDWORK, A( I+IB, I ), LDA, ONE,
$              A( 1, I+IB ), LDA )
          A( I+IB, I+IB-1 ) = EI
*
*      Apply the block reflector H to A(1:i,i+1:i+ib-1) from the
*      right
*
          CALL ZTRMM( 'Right', 'Lower', 'Conjugate transpose',
$              'Unit', I, IB-1,
$              ONE, A( I+1, I ), LDA, WORK, LDWORK )
          DO 30 J = 0, IB-2
              CALL ZAXPY( I, -ONE, WORK( LDWORK*J+1 ), 1,
$                  A( 1, I+J+1 ), 1 )
30      CONTINUE
*
*      Apply the block reflector H to A(i+1:ihi,i+ib:n) from the
*      left
*
          CALL ZLARFB( 'Left', 'Conjugate transpose', 'Forward',
$              'Columnwise',
$              IHI-I, N-I-IB+1, IB, A( I+1, I ), LDA, T, LDT,
$              A( I+1, I+IB ), LDA, WORK, LDWORK )
40      CONTINUE
      END IF
*
*      Use unblocked code to reduce the rest of the matrix
*
      CALL ZGEHD2( N, I, IHI, A, LDA, TAU, WORK, IINFO )
      WORK( 1 ) = IWS
*

```

```

        RETURN
*
*      End of ZGEHRD
*
      END

```

— LAPACK zgehrd —

```

(let*
  ((nbmax 64) (ldt (+ nbmax 1))
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:integer4 64 64) nbmax) (type (f2cl-lib:integer4) ldt)
            (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
            (ignorable nbmax ldt zero one))
  (defun zgehrd (n ilo ihi a lda tau work lwork info)
    (declare (type (f2cl-lib:integer4) info lwork lda ihi ilo n)
              (type (array f2cl-lib:complex16 (*)) work tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (prog
        ((ei #C(0.0d0 0.0d0)) (i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ldwork 0)
         (lwko 0) (nb 0) (nbmin 0) (nh 0) (nx 0) (lquery nil)
         (t$
          (make-array (the fixnum (reduce #'* (list ldt nbmax))) :element-type
                       'f2cl-lib:complex16)))
        (declare (type (f2cl-lib:complex16) ei)
                  (type (f2cl-lib:integer4)
                        nx nh nbmin nb lwko ldwork j iws iinfo ib i)
                  (type f2cl-lib:logical lquery)
                  (type (array f2cl-lib:complex16 (*)) t$))
        (setf info 0)
        (setf nb
          (min (the f2cl-lib:integer4 nbmax)
               (the f2cl-lib:integer4
                 (multiple-value-bind
                   (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                   (ilaenv 1 "ZGEHRD" " " n ilo ihi -1)
                   (declare (ignore var-0 var-1 var-2 var-6))
                   (when var-3 (setf n var-3))
                   (when var-4 (setf ilo var-4))
                   (when var-5 (setf ihi var-5)) ret-val))))
        (setf lwko (f2cl-lib:int-mul n nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)

```

```

(coerce lwkopt 'f2cl-lib:complex16))
(setf lquery (coerce (= lwork -1) 'f2cl-lib:logical))
(cond ((< n 0) (setf info -1))
      ((or (< ilo 1)
            (> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))
       (setf info -2))
      ((or (< ihi (min (the f2cl-lib:integer4 ilo)
                       (the f2cl-lib:integer4 n)))
            (> ihi n))
       (setf info -3))
      (< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
      (setf info -5))
      ((and (< lwork (max (the f2cl-lib:integer4 1)
                          (the f2cl-lib:integer4 n)))
            (not lquery))
       (setf info -8)))
(cond ((/= info 0)
      (xerbla "ZGEHRD" (f2cl-lib:int-sub info))
      (go end_label))
      (lquery (go end_label)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
  ((> i
    (f2cl-lib:int-add ilo (f2cl-lib:int-sub 1)))
   nil)
  (tagbody
   (setf (f2cl-lib:fref tau-%data% (i) ((1 *))
         tau-%offset%) zero) label10)
  )
(f2cl-lib:fdo (i (max (the f2cl-lib:integer4 1)
                     (the f2cl-lib:integer4 ihi))
              (f2cl-lib:int-add i 1))
  ((> i (f2cl-lib:int-add n (f2cl-lib:int-sub 1))) nil)
  (tagbody
   (setf (f2cl-lib:fref tau-%data% (i) ((1 *))
         tau-%offset%) zero) label20)
  )
)
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(cond
  ((<= nh 1)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce 1 'f2cl-lib:complex16))
   (go end_label)))
(setf nb
  (min (the f2cl-lib:integer4 nbmax)
        (the f2cl-lib:integer4
          (multiple-value-bind (ret-val var-0 var-1 var-2 var-3 var-4
                              var-5 var-6)
            (ilaenv 1 "ZGEHRD" " " n ilo ihi -1)
            (declare (ignore var-0 var-1 var-2 var-6))
            (when var-3 (setf n var-3))

```

```

      (when var-4 (setf ilo var-4))
      (when var-5 (setf ihi var-5)) ret-val))))
(setf nbmin 2) (setf iws 1)
(cond
  ((and (> nb 1) (< nb nh))
   (setf nx
    (max (the f2cl-lib:integer4 nb)
         (the f2cl-lib:integer4
          (multiple-value-bind (ret-val var-0 var-1 var-2 var-3 var-4
                               var-5 var-6)
            (ilaenv 3 "ZGEHRD" " " n ilo ihi -1)
            (declare (ignore var-0 var-1 var-2 var-6))
            (when var-3 (setf n var-3))
            (when var-4 (setf ilo var-4))
            (when var-5 (setf ihi var-5)) ret-val))))))
  (cond
    ((< nx nh) (setf iws (f2cl-lib:int-mul n nb))
     (cond
       ((< lwork iws)
        (setf nbmin
         (max (the f2cl-lib:integer4 2)
              (the f2cl-lib:integer4
               (multiple-value-bind
                 (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                 (ilaenv 2 "ZGEHRD" " " n ilo ihi -1)
                 (declare (ignore var-0 var-1 var-2 var-6))
                 (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
                 (when var-5 (setf ihi var-5)) ret-val))))))
        (cond
          ((>= lwork (f2cl-lib:int-mul n nbmin))
           (setf nb (the f2cl-lib:integer4 (truncate lwork n))))
          (t (setf nb 1)))))))
    (setf ldwork n)
    (cond ((or (< nb nbmin) (>= nb nh)) (setf i ilo))
    (t
     (f2cl-lib:fdo (i ilo (f2cl-lib:int-add i nb))
       (> i
        (f2cl-lib:int-add ihi
         (f2cl-lib:int-sub 1) (f2cl-lib:int-sub nx)))
       nil)
     (tagbody
      (setf ib
       (min (the f2cl-lib:integer4 nb)
            (the f2cl-lib:integer4 (f2cl-lib:int-sub ihi i))))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9)
        (zlahr2 ihi i ib
         (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
          (1 i) ((1 lda) (1 *)))

```

```

a-%offset%)
lda
(f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
(i) ((1 *))
tau-%offset%)
t$ ldt work ldwork)
(declare (ignore var-3 var-5 var-6 var-8))
(when var-0 (setf ihi var-0))
(when var-1 (setf i var-1)) (when var-2 (setf ib var-2))
(when var-4 (setf lda var-4))
(when var-7 (setf ldt var-7))
(when var-9 (setf ldwork var-9)))
(setf ei
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add i ib)
(f2cl-lib:int-sub (f2cl-lib:int-add i ib) 1))
((1 lda) (1 *)) a-%offset%))
(setf
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add i ib)
(f2cl-lib:int-sub (f2cl-lib:int-add i ib) 1))
((1 lda) (1 *)) a-%offset%)
one)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10 var-11
var-12)
(zgemm "No transpose" "Conjugate transpose" ihi
(f2cl-lib:int-add (f2cl-lib:int-sub ihi i ib) 1)
ib (- one) work ldwork
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
((+ i ib) i)
((1 lda) (1 *)) a-%offset%)
lda one
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
(1 (f2cl-lib:int-add i ib)) ((1 lda) (1 *))
a-%offset%)
lda)
(declare (ignore var-0 var-1 var-3 var-5 var-6
var-8 var-11))
(when var-2 (setf ihi var-2))
(when var-4 (setf ib var-4))
(when var-7 (setf ldwork var-7))
(when var-9 (setf lda var-9))
(when var-10 (setf one var-10))
(when var-12 (setf lda var-12)))
(setf
(f2cl-lib:fref a-%data%
((f2cl-lib:int-add i ib)
(f2cl-lib:int-sub (f2cl-lib:int-add i ib) 1))

```

```

      ((1 lda) (1 *)) a-%offset%)
    ei)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10)
    (ztrmm "Right" "Lower" "Conjugate transpose" "Unit" i
      (f2cl-lib:int-sub ib 1) one
      (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
        ((+ i 1) i)
        ((1 lda) (1 *)) a-%offset%)
      lda work ldwork)
    (declare (ignore var-0 var-1 var-2 var-3 var-5
      var-7 var-9))
    (when var-4 (setf i var-4))
    (when var-6 (setf one var-6))
    (when var-8 (setf lda var-8))
    (when var-10 (setf ldwork var-10)))
    (f2cl-lib:fdo (j 0 (f2cl-lib:int-add j 1))
      (> j
        (f2cl-lib:int-add ib (f2cl-lib:int-sub 2)))
        nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4
      var-5)
      (zaxpy i (- one)
        (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
          ((+ (f2cl-lib:int-mul ldwork j) 1)) ((1 *))
          work-%offset%)
          1
          (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
            (1 (f2cl-lib:int-add i j 1)) ((1 lda) (1 *))
            a-%offset%)
            1)
      (declare (ignore var-1 var-2 var-3 var-4 var-5))
      (when var-0 (setf i var-0)))
      label30))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10 var-11
     var-12 var-13 var-14)
    (zlarfb "Left" "Conjugate transpose"
      "Forward" "Columnwise"
      (f2cl-lib:int-sub ihi i)
      (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1)
      ib
      (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
        ((+ i 1) i)
        ((1 lda) (1 *)) a-%offset%)
      lda t$ ldt
      (f2cl-lib:array-slice a-%data% f2cl-lib:complex16

```

```

      ((+ i 1) (f2cl-lib:int-add i ib)) ((1 lda) (1 *))
      a-%offset%)
      lda work ldwork)
      (declare
        (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7
          var-9 var-11 var-13))
      (when var-6 (setf ib var-6))
      (when var-8 (setf lda var-8))
      (when var-10 (setf ldt var-10))
      (when var-12 (setf lda var-12))
      (when var-14 (setf ldwork var-14)))
      label40)))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
        (zgehd2 n i ihi a lda tau work iinfo)
        (declare (ignore var-0 var-1 var-3 var-5 var-6)) (setf ihi var-2)
        (setf lda var-4) (setf iinfo var-7))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce iws 'f2cl-lib:complex16))
        (go end_label)
      end_label
      (return (values n ilo ihi nil lda nil nil nil info)))
    )))

```

zhseqr LAPACK

— zhseqr.input —

```

)set break resume
)sys rm -f zhseqr.output
)spool zhseqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zhseqr.help —

```

=====
zhseqr examples
=====

```



```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

```
Definition:
=====
```

```
SUBROUTINE ZHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, W, Z, LDZ,
                  WORK, LWORK, INFO )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHI, ILO, INFO, LDH, LDZ, LWORK, N
CHARACTER        COMPZ, JOB
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
..
```

```
Purpose:
=====
```

ZHSEQR computes the eigenvalues of a Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z T Z^*H$, where T is an upper triangular matrix (the Schur form), and Z is the unitary matrix of Schur vectors.

Optionally Z may be postmultiplied into an input unitary matrix Q so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the unitary matrix Q: $A = Q^*H^*Q^*H = (QZ)^*H^*(QZ)^*H$.

```
Arguments:
=====
```

[in] JOB

```
JOB is CHARACTER*1
= 'E': compute eigenvalues only;
= 'S': compute eigenvalues and the Schur form T.
```

[in] COMPZ

```
COMPZ is CHARACTER*1
= 'N': no Schur vectors are computed;
```

= 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned;
 = 'V': Z must contain an unitary matrix Q on entry, and the product $Q*Z$ is returned.

[in] N

N is INTEGER
 The order of the matrix H. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by ZGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively. If $N > 0$, then $1 \leq \text{ILO} \leq \text{IHI} \leq N$. If $N = 0$, then $\text{ILO} = 1$ and $\text{IHI} = 0$.

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)
 On entry, the upper Hessenberg matrix H.
 On exit, if INFO = 0 and JOB = 'S', H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit. (The output value of H when $\text{INFO} > 0$ is given under the description of INFO below.)

Unlike earlier versions of ZHSEQR, this subroutine may explicitly $H(i,j) = 0$ for $i > j$ and $j = 1, 2, \dots, \text{ILO}-1$ or $j = \text{IHI}+1, \text{IHI}+2, \dots, N$.

[in] LDH

LDH is INTEGER
 The leading dimension of the array H. $\text{LDH} \geq \max(1, N)$.

[out] W

W is COMPLEX*16 array, dimension (N)
 The computed eigenvalues. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur

form returned in H, with $W(i) = H(i,i)$.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,N)
 If COMPZ = 'N', Z is not referenced.
 If COMPZ = 'I', on entry Z need not be set and on exit,
 if INFO = 0, Z contains the unitary matrix Z of the Schur
 vectors of H. If COMPZ = 'V', on entry Z must contain an
 N-by-N matrix Q, which is assumed to be equal to the unit
 matrix except for the submatrix Z(ILO:IHI,ILO:IHI). On exit,
 if INFO = 0, Z contains $Q*Z$.
 Normally Q is the unitary matrix generated by ZUNGHR
 after the call to ZGEHRD which formed the Hessenberg matrix
 H. (The output value of Z when INFO.GT.0 is given under
 the description of INFO below.)

[in] LDZ

LDZ is INTEGER
 The leading dimension of the array Z. if COMPZ = 'I' or
 COMPZ = 'V', then LDZ.GE.MAX(1,N). Otherwise, LDZ.GE.1.

[out] WORK

WORK is COMPLEX*16 array, dimension (LWORK)
 On exit, if INFO = 0, WORK(1) returns an estimate of
 the optimal value for LWORK.

[in] LWORK

LWORK is INTEGER
 The dimension of the array WORK. LWORK .GE. max(1,N)
 is sufficient and delivers very good and sometimes
 optimal performance. However, LWORK as large as 11*N
 may be required for optimal performance. A workspace
 query is recommended to determine the optimal workspace
 size.

If LWORK = -1, then ZHSEQR does a workspace query.
 In this case, ZHSEQR checks the input parameters and
 estimates the optimal workspace size for the given
 values of N, ILO and IHI. The estimate is returned
 in WORK(1). No error message related to LWORK is
 issued by XERBLA. Neither H nor Z are accessed.

[out] INFO

INFO is INTEGER
 = 0: successful exit

.LT. 0: if INFO = -i, the i-th argument had an illegal value

.GT. 0: if INFO = i, ZHSEQR failed to compute all of the eigenvalues. Elements 1:i-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and JOB = 'E', then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and JOB = 'S', then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and COMPZ = 'V', then on exit

(final value of Z) = (initial value of Z)*U

where U is the unitary matrix in (*) (regardless of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'I', then on exit
(final value of Z) = U

where U is the unitary matrix in (*) (regardless of the value of JOB.)

If INFO .GT. 0 and COMPZ = 'N', then Z is not accessed.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Contributors:
=====

Karen Braman and Ralph Byers, Department of Mathematics,
University of Kansas, USA

Further Details:

=====

Default values supplied by
ILAENV(ISPEC,'ZHSEQR',JOB(:1)//COMPZ(:1),N,ILO,IHI,LWORK).
It is suggested that these defaults be adjusted in order
to attain best performance in each particular
computational environment.

ISPEC=12: The ZLAHQR vs ZLAQRO crossover point.
Default: 75. (Must be at least 11.)

ISPEC=13: Recommended deflation window size.
This depends on ILO, IHI and NS. NS is the
number of simultaneous shifts returned
by ILAENV(ISPEC=15). (See ISPEC=15 below.)
The default for (IHI-ILO+1).LE.500 is NS.
The default for (IHI-ILO+1).GT.500 is 3*NS/2.

ISPEC=14: Nibble crossover point. (See IPARMQ for
details.) Default: 14% of deflation window
size.

ISPEC=15: Number of simultaneous shifts in a multishift
QR iteration.

If IHI-ILO+1 is ...

| greater than or equal to ... | ...but less than | ... the default is |
|---------------------------------|---------------------|-----------------------|
| 1 | 30 | NS = 2(+) |
| 30 | 60 | NS = 4(+) |
| 60 | 150 | NS = 10(+) |
| 150 | 590 | NS = ** |
| 590 | 3000 | NS = 64 |
| 3000 | 6000 | NS = 128 |
| 6000 | infinity | NS = 256 |

(+) By default some or all matrices of this order
are passed to the implicit double shift routine
ZLAHQR and this parameter is ignored. See
ISPEC=12 above and comments in IPARMQ for
details.

(**) The asterisks (**) indicate an ad-hoc
function of N increasing from 10 to 64.

ISPEC=16: Select structured matrix multiply.
 If the number of simultaneous shifts (specified
 by ISPEC=15) is less than 14, then the default
 for ISPEC=16 is 0. Otherwise the default for
 ISPEC=16 is 2.

References:

=====

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR
 Algorithm Part I: Maintaining Well Focused Shifts, and Level 3
 Performance, SIAM Journal of Matrix Analysis, volume 23, pages
 929--947, 2002.

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR
 Algorithm Part II: Aggressive Early Deflation, SIAM Journal
 of Matrix Analysis, volume 23, pages 948--973, 2002.

— zhseqr.f —

```
* =====
*      SUBROUTINE ZHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, W, Z, LDZ,
*      $                WORK, LWORK, INFO )
*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, ILO, INFO, LDH, LDZ, LWORK, N
*      CHARACTER        COMPZ, JOB
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
*
*      ..
*
* =====
*
*      .. Parameters ..
*
*      ==== Matrices of order NTINY or smaller must be processed by
*      .    ZLAHQR because of insufficient subdiagonal scratch space.
*      .    (This is a hard limit.) ====
*      INTEGER          NTINY
```

```

      PARAMETER          ( NTINY = 11 )

*
*  ==== NL allocates some local workspace to help small matrices
*  .   through a rare ZLAHQR failure.  NL .GT. NTINY = 11 is
*  .   required and NL .LE. NMIN = ILAENV(ISPEC=12,...) is recom-
*  .   mended.  (The default value of NMIN is 75.)  Using NL = 49
*  .   allows up to six simultaneous shifts and a 16-by-16
*  .   deflation window.  ====
      INTEGER            NL
      PARAMETER          ( NL = 49 )
      COMPLEX*16          ZERO, ONE
      PARAMETER          ( ZERO = ( 0.0d0, 0.0d0 ),
$      ONE = ( 1.0d0, 0.0d0 ) )
      DOUBLE PRECISION    RZERO
      PARAMETER          ( RZERO = 0.0d0 )

*
*  ..
*  .. Local Arrays ..
      COMPLEX*16          HL( NL, NL ), WORKL( NL )
*
*  ..
*  .. Local Scalars ..
      INTEGER            KBOT, NMIN
      LOGICAL            INITZ, LQUERY, WANTT, WANTZ
*
*  ..
*  .. External Functions ..
      INTEGER            ILAENV
      LOGICAL            LSAME
      EXTERNAL           ILAENV, LSAME
*
*  ..
*  .. External Subroutines ..
      EXTERNAL           XERBLA, ZCOPY, ZLACPY, ZLAHQR, ZLAQRO, ZLASET
*
*  ..
*  .. Intrinsic Functions ..
      INTRINSIC          DBLE, DCMLPX, MAX, MIN
*
*  ..
*  .. Executable Statements ..
*
*  ==== Decode and check the input parameters. ====
*
      WANTT = LSAME( JOB, 'S' )
      INITZ = LSAME( COMPZ, 'I' )
      WANTZ = INITZ .OR. LSAME( COMPZ, 'V' )
      WORK( 1 ) = DCMLPX( DBLE( MAX( 1, N ) ), RZERO )
      LQUERY = LWORK.EQ.-1
*
      INFO = 0
      IF( .NOT.LSAME( JOB, 'E' ) .AND. .NOT.WANTT ) THEN
         INFO = -1
      ELSE IF( .NOT.LSAME( COMPZ, 'N' ) .AND. .NOT.WANTZ ) THEN
         INFO = -2
      ELSE IF( N.LT.0 ) THEN

```

```

        INFO = -3
    ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
        INFO = -4
    ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -5
    ELSE IF( LDH.LT.MAX( 1, N ) ) THEN
        INFO = -7
    ELSE IF( LDZ.LT.1 .OR. ( WANTZ .AND. LDZ.LT.MAX( 1, N ) ) ) THEN
        INFO = -10
    ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
        INFO = -12
    END IF

*
    IF( INFO.NE.0 ) THEN
*
*       ==== Quick return in case of invalid argument. ====
*
        CALL XERBLA( 'ZHSEQR', -INFO )
        RETURN
*
    ELSE IF( N.EQ.0 ) THEN
*
*       ==== Quick return in case N = 0; nothing to do. ====
*
        RETURN
*
    ELSE IF( LQUERY ) THEN
*
*       ==== Quick return in case of a workspace query ====
*
        CALL ZLAQRO( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILO, IHI, Z,
$           LDZ, WORK, LWORK, INFO )
*       ==== Ensure reported workspace size is backward-compatible with
*       .   previous LAPACK versions. ====
        WORK( 1 ) = DCMPLX( MAX( DBLE( WORK( 1 ) ), DBLE( MAX( 1,
$           N ) ) ), RZERO )
        RETURN
*
    ELSE
*
*       ==== copy eigenvalues isolated by ZGEBAL ====
*
        IF( ILO.GT.1 )
$           CALL ZCOPY( ILO-1, H, LDH+1, W, 1 )
        IF( IHI.LT.N )
$           CALL ZCOPY( N-IHI, H( IHI+1, IHI+1 ), LDH+1, W( IHI+1 ), 1 )
*
*       ==== Initialize Z, if requested ====
*
        IF( INITZ )

```



```

$      CALL ZLASET( 'A', N, N, ZERO, ONE, Z, LDZ )
*
*      ==== Quick return if possible ====
*
      IF( ILO.EQ.IHI ) THEN
        W( ILO ) = H( ILO, ILO )
        RETURN
      END IF
*
*      ==== ZLAHQR/ZLAQRO crossover point ====
*
      NMIN = ILAENV( 12, 'ZHSEQR', JOB // COMPZ, N,
$      ILO, IHI, LWORK )
      NMIN = MAX( NTINY, NMIN )
*
*      ==== ZLAQRO for big matrices; ZLAHQR for small ones ====
*
      IF( N.GT.NMIN ) THEN
        CALL ZLAQRO( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILO, IHI,
$      Z, LDZ, WORK, LWORK, INFO )
      ELSE
*
*      ==== Small matrix ====
*
        CALL ZLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILO, IHI,
$      Z, LDZ, INFO )
*
        IF( INFO.GT.0 ) THEN
*
*      ==== A rare ZLAHQR failure! ZLAQRO sometimes succeeds
*      .   when ZLAHQR fails. ====
*
          KBOT = INFO
*
          IF( N.GE.NL ) THEN
*
*      ==== Larger matrices have enough subdiagonal scratch
*      .   space to call ZLAQRO directly. ====
*
            CALL ZLAQRO( WANTT, WANTZ, N, ILO, KBOT, H, LDH, W,
$      ILO, IHI, Z, LDZ, WORK, LWORK, INFO )
*
          ELSE
*
*      ==== Tiny matrices don't have enough subdiagonal
*      .   scratch space to benefit from ZLAQRO. Hence,
*      .   tiny matrices must be copied into a larger
*      .   array before calling ZLAQRO. ====
*
            CALL ZLACPY( 'A', N, N, H, LDH, HL, NL )

```

```

        HL( N+1, N ) = ZERO
        CALL ZLASET( 'A', NL, NL-N, ZERO, ZERO, HL( 1, N+1 ),
$              NL )
        CALL ZLAQRO( WANTT, WANTZ, NL, ILO, KBOT, HL, NL, W,
$              ILO, IHI, Z, LDZ, WORKL, NL, INFO )
        IF( WANTT .OR. INFO.NE.0 )
$              CALL ZLACPY( 'A', N, N, HL, NL, H, LDH )
        END IF
    END IF
END IF

*
*      ==== Clear out the trash, if necessary. ====
*
    IF( ( WANTT .OR. INFO.NE.0 ) .AND. N.GT.2 )
$        CALL ZLASET( 'L', N-2, N-2, ZERO, ZERO, H( 3, 1 ), LDH )
*
*      ==== Ensure reported workspace size is backward-compatible with
*      .      previous LAPACK versions. ====
*
    WORK( 1 ) = DCMPLX( MAX( DBLE( MAX( 1, N ) ),
$        DBLE( WORK( 1 ) ) ), RZERO )
    END IF

*
*      ==== End of ZHSEQR ====
*
END

```

— LAPACK zhseqr —

```

(let*
  ((ntiny 11) (nl 49)
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16))
   (rzero 0.0d0))
  (declare (type (f2cl-lib:integer4 11 11) ntiny)
            (type (f2cl-lib:integer4 49 49) nl) (type (f2cl-lib:complex16) zero)
            (type (f2cl-lib:complex16) one) (type (double-float 0.0d0 0.0d0) rzero)
            (ignorable ntiny nl zero one rzero))
  (defun zhseqr (job compz n ilo ihi h ldh w z ldz work lwork info)
    (declare (type (simple-array character (*)) compz job)
              (type (f2cl-lib:integer4) info lwork ldz ldh ihi ilo n)
              (type (array f2cl-lib:complex16 (*)) work z w h))
    (f2cl-lib:with-multi-array-data
      ((h f2cl-lib:complex16 h-%data% h-%offset%)
       (w f2cl-lib:complex16 w-%data% w-%offset%)
       (z f2cl-lib:complex16 z-%data% z-%offset%))

```

```

(work f2cl-lib:complex16 work-%data% work-%offset%)
(job character job-%data% job-%offset%)
(compz character compz-%data% compz-%offset%))
(prog
  ((initz nil) (lquery nil) (wantt nil) (wantz nil) (kbot 0) (nmin 0)
   (hl
    (make-array (the fixnum (reduce #'* (list nl nl))) :element-type
      'f2cl-lib:complex16))
    (workl (make-array nl :element-type 'f2cl-lib:complex16)))
  (declare (type f2cl-lib:logical wantz wantt lquery initz)
    (type (f2cl-lib:integer4) nmin kbot)
    (type (array f2cl-lib:complex16 (*)) workl hl))
  (setf wantt
    (multiple-value-bind (ret-val var-0 var-1) (lsame job "S")
      (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
  (setf initz
    (multiple-value-bind (ret-val var-0 var-1) (lsame compz "I")
      (declare (ignore var-1)) (when var-0 (setf compz var-0)) ret-val))
  (setf wantz
    (or initz
      (multiple-value-bind (ret-val var-0 var-1) (lsame compz "V")
        (declare (ignore var-1)) (when var-0 (setf compz var-0)) ret-val)))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (f2cl-lib:dcmplx
      (f2cl-lib:dbple (max (the f2cl-lib:integer4 1)
        (the f2cl-lib:integer4 n)))
      rzero))
  (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical)) (setf info 0)
  (cond
    ((and
      (not
        (multiple-value-bind (ret-val var-0 var-1) (lsame job "E")
          (declare (ignore var-1)) (when var-0 (setf job var-0)) ret-val))
        (not wantt))
      (setf info -1))
    ((and
      (not
        (multiple-value-bind (ret-val var-0 var-1) (lsame compz "N")
          (declare (ignore var-1)) (when var-0 (setf compz var-0)) ret-val))
        (not wantz))
      (setf info -2))
    ((< n 0) (setf info -3))
    ((or (< ilo 1)
      (> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))
      (setf info -4))
    ((or (< ihi (min (the f2cl-lib:integer4 ilo)
      (the f2cl-lib:integer4 n)))
      (> ihi n))
      (setf info -5))
    ((< ldh (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))

```

```

(setf info -7))
((or (< ldz 1)
    (and wantz
        (< ldz (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))))
(setf info -10))
((and (< lwork (max (the f2cl-lib:integer4 1)
                    (the f2cl-lib:integer4 n)))
    (not lquery))
 (setf info -12)))
(cond ((/= info 0)
      (xerbla "ZHSEQR" (f2cl-lib:int-sub info))
      (go end_label))
      ((= n 0) (go end_label))
      (lquery
       (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
         var-10 var-11
         var-12 var-13 var-14)
        (zlaqr0 wantt wantz n ilo ihi h ldh w ilo ihi z ldz work lwork info)
        (declare (ignore var-5 var-7 var-10 var-12))
        (when var-0 (setf wantt var-0)) (when var-1 (setf wantz var-1))
        (when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
        (when var-4 (setf ihi var-4)) (when var-6 (setf ldh var-6))
        (when var-8 (setf ilo var-8)) (when var-9 (setf ihi var-9))
        (when var-11 (setf ldz var-11)) (when var-13 (setf lwork var-13))
        (when var-14 (setf info var-14)))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
              (f2cl-lib:dcmplx
               (max (f2cl-lib:dbple (f2cl-lib:fref work-%data% (1) ((1 *))
                                   work-%offset%))
                    (f2cl-lib:dbple
                     (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))
              rzero))
        (go end_label))
      (t
       (if (> ilo 1)
           (zcopy (f2cl-lib:int-sub ilo 1) h (f2cl-lib:int-add ldh 1) w 1))
       (if (< ihi n)
           (zcopy (f2cl-lib:int-sub n ihi)
                  (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
                    ((+ ihi 1) (f2cl-lib:int-add ihi 1)) ((1 ldh) (1 *)) h-%offset%)
                  (f2cl-lib:int-add ldh 1)
                  (f2cl-lib:array-slice w-%data% f2cl-lib:complex16
                    ((+ ihi 1)) ((1 *))
                    w-%offset%)
                  1))
       (if initz
           (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
            (zlaset "A" n n zero one z ldz) (declare (ignore var-0 var-5))
            (when var-1 (setf n var-1)) (when var-2 (setf n var-2))

```

```

(when var-3 (setf zero var-3)) (when var-4 (setf one var-4))
(when var-6 (setf ldz var-6)))
(cond
  ((= ilo ihi)
    (setf (f2cl-lib:fref w-%data% (ilo) ((1 *)) w-%offset%)
      (f2cl-lib:fref h-%data% (ilo ilo) ((1 ldh) (1 *)) h-%offset%))
    (go end_label)))
(setf nmin
  (multiple-value-bind (ret-val var-0 var-1 var-2 var-3 var-4
    var-5 var-6)
    (ilaenv 12 "ZHSEQR" (f2cl-lib:f2cl-// job compz) n ilo ihi lwork)
    (declare (ignore var-0 var-1 var-2)) (when var-3 (setf n var-3))
    (when var-4 (setf ilo var-4)) (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nmin (max (the f2cl-lib:integer4 ntiny)
  (the f2cl-lib:integer4 nmin)))
(cond
  ((> n nmin)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
        var-10
        var-11 var-12 var-13 var-14)
      (zlaqr0 wantt wantz n ilo ihi h ldh w ilo ihi z
        ldz lwork lwork info)
      (declare (ignore var-5 var-7 var-10 var-12))
      (when var-0 (setf wantt var-0)) (when var-1 (setf wantz var-1))
      (when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
      (when var-4 (setf ihi var-4)) (when var-6 (setf ldh var-6))
      (when var-8 (setf ilo var-8)) (when var-9 (setf ihi var-9))
      (when var-11 (setf ldz var-11)) (when var-13 (setf lwork var-13))
      (when var-14 (setf info var-14))))
  (t
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
        var-10
        var-11 var-12)
      (zlahqr wantt wantz n ilo ihi h ldh w ilo ihi z ldz info)
      (declare (ignore var-5 var-7 var-10))
      (when var-0 (setf wantt var-0))
      (when var-1 (setf wantz var-1)) (when var-2 (setf n var-2))
      (when var-3 (setf ilo var-3)) (when var-4 (setf ihi var-4))
      (when var-6 (setf ldh var-6)) (when var-8 (setf ilo var-8))
      (when var-9 (setf ihi var-9)) (when var-11 (setf ldz var-11))
      (when var-12 (setf info var-12)))
    (cond
      ((> info 0) (setf kbot info)
        (cond
          ((>= n nl)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8

```

```

var-9 var-10
var-11 var-12 var-13 var-14)
(zlaqr0 wantt wantz n ilo kbot h ldh w ilo ihi z
  ldz work lwork info)
(declare (ignore var-5 var-7 var-10 var-12))
(when var-0 (setf wantt var-0))
(when var-1 (setf wantz var-1))
(when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
(when var-4 (setf kbot var-4)) (when var-6 (setf ldh var-6))
(when var-8 (setf ilo var-8)) (when var-9 (setf ihi var-9))
(when var-11 (setf ldz var-11))
(when var-13 (setf lwork var-13))
(when var-14 (setf info var-14)))
(t
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (zlacpy "A" n n h ldh hl nl)
  (declare (ignore var-0 var-3 var-5))
  (when var-1 (setf n var-1)) (when var-2 (setf n var-2))
  (when var-4 (setf ldh var-4)) (when var-6 (setf nl var-6)))
(setf (f2cl-lib:fref hl ((f2cl-lib:int-add n 1) n)
      ((1 nl) (1 nl)))
zero)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (zlaset "A" nl (f2cl-lib:int-sub nl n) zero zero
    (f2cl-lib:array-slice hl f2cl-lib:complex16
      (1 (f2cl-lib:int-add n 1)) ((1 nl) (1 nl)))
    nl)
  (declare (ignore var-0 var-2 var-5))
  (when var-1 (setf nl var-1))
  (when var-3 (setf zero var-3)) (when var-4 (setf zero var-4))
  (when var-6 (setf nl var-6)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10
   var-11 var-12 var-13 var-14)
  (zlaqr0 wantt wantz nl ilo kbot hl nl w ilo ihi z
    ldz workl nl info)
  (declare (ignore var-5 var-7 var-10 var-12))
  (when var-0 (setf wantt var-0))
  (when var-1 (setf wantz var-1))
  (when var-2 (setf nl var-2)) (when var-3 (setf ilo var-3))
  (when var-4 (setf kbot var-4)) (when var-6 (setf nl var-6))
  (when var-8 (setf ilo var-8)) (when var-9 (setf ihi var-9))
  (when var-11 (setf ldz var-11)) (when var-13 (setf nl var-13))
  (when var-14 (setf info var-14)))
(if (or wantt (/= info 0))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6)

```

```

      (zlacpy "A" n n hl nl h ldh)
      (declare (ignore var-0 var-3 var-5))
      (when var-1 (setf n var-1))
      (when var-2 (setf n var-2))
      (when var-4 (setf nl var-4))
      (when var-6 (setf ldh var-6)))))))))
  (if (and (or wantt (/= info 0)) (> n 2))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
        (zlaset "L" (f2cl-lib:int-sub n 2) (f2cl-lib:int-sub n 2) zero zero
          (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
            (3 1) ((1 ldh) (1 *)))
            h-%offset%)
          ldh)
        (declare (ignore var-0 var-1 var-2 var-5))
        (when var-3 (setf zero var-3))
        (when var-4 (setf zero var-4))
        (when var-6 (setf ldh var-6))))
      (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (f2cl-lib:dcmplx
          (max
            (f2cl-lib:dbple
              (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
            (f2cl-lib:dbple
              (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)))
          rzero))))
  end_label
  (return
    (values job compz n ilo ihi nil ldh nil nil ldz nil lwork info)))
)))

```

zlacgv LAPACK

— zlacgv.input —

```

)set break resume
)sys rm -f zlacgv.output
)spool zlacgv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlacgv.help —

```
=====
zlacgv examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====

SUBROUTINE ZLACGV( N, X, INCX )

  .. Scalar Arguments ..
  INTEGER          INCX, N
  ..
  .. Array Arguments ..
  COMPLEX*16       X( * )
  ..
```

Purpose:

```
=====
```

ZLACGV conjugates a complex vector of length N.

Arguments:

```
=====
```

[in] N

N is INTEGER
 The length of the vector X. N >= 0.

[in,out] X

X is COMPLEX*16 array, dimension
 (1+(N-1)*abs(INCX))
 On entry, the vector of length N to be conjugated.
 On exit, X is overwritten with conjg(X).

[in] INCX

INCX is INTEGER

The spacing between successive elements of X.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlacgv.f —

```
* =====
*      SUBROUTINE ZLACGV( N, X, INCX )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          INCX, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       X( * )
*
*      ..
*
* =====
*
*      .. Local Scalars ..
*      INTEGER          I, IOFF
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC        DCONJG
*
*      ..
*      .. Executable Statements ..
*
*      IF( INCX.EQ.1 ) THEN
*          DO 10 I = 1, N
*              X( I ) = DCONJG( X( I ) )
* 10      CONTINUE
*      ELSE
*          IOFF = 1
```

```

      IF( INCX.LT.0 )
$      IOFF = 1 - ( N-1 )*INCX
      DO 20 I = 1, N
          X( IOFF ) = DCONJG( X( IOFF ) )
          IOFF = IOFF + INCX
20      CONTINUE
      END IF
      RETURN
*
*      End of ZLAGV
*
      END

```

— LAPACK zlagv —

```

(defun zlagv (n x incx)
  (declare (type (f2cl-lib:integer4) incx n)
    (type (array f2cl-lib:complex16 (*)) x))
  (f2cl-lib:with-multi-array-data
    ((x f2cl-lib:complex16 x-%data% x-%offset%))
    (prog
      ((i 0) (ioff 0)) (declare (type (f2cl-lib:integer4) ioff i))
      (cond
        ((= incx 1)
          (f2cl-lib:fdof (i 1 (f2cl-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%)
                (coerce (f2cl-lib:dconjg
                  (f2cl-lib:fref x-%data% (i) ((1 *)) x-%offset%))
                  'f2cl-lib:complex16))
              label10))))
        (t (setf ioff 1)
          (if (< incx 0)
            (setf ioff
              (f2cl-lib:int-sub 1
                (f2cl-lib:int-mul (f2cl-lib:int-sub n 1) incx))))
          (f2cl-lib:fdof (i 1 (f2cl-lib:int-add i 1))
            ((> i n) nil)
            (tagbody
              (setf (f2cl-lib:fref x-%data% (ioff) ((1 *)) x-%offset%)
                (coerce
                  (f2cl-lib:dconjg (f2cl-lib:fref x-%data% (ioff)
                    ((1 *)) x-%offset%))
                  'f2cl-lib:complex16))
              (setf ioff (f2cl-lib:int-add ioff incx)) label20))))))

```

```
(go end_label) end_label (return (values nil nil nil))))))
```

zlacpy LAPACK

— zlacpy.input —

```
)set break resume
)sys rm -f zlacpy.output
)spool zlacpy.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zlacpy.help —

```
=====
zlacpy examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLACPY( UPLO, M, N, A, LDA, B, LDB )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          UPLO
```

```
INTEGER            LDA, LDB, M, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16         A( LDA, * ), B( LDB, * )
```

```
..
```

Purpose:

=====

ZLACPY copies all or part of a two-dimensional matrix A to another matrix B.

Arguments:

=====

[in] UPLO

UPLO is CHARACTER*1

Specifies the part of the matrix A to be copied to B.

= 'U': Upper triangular part

= 'L': Lower triangular part

Otherwise: All of the matrix A

[in] M

M is INTEGER

The number of rows of the matrix A. $M \geq 0$.

[in] N

N is INTEGER

The number of columns of the matrix A. $N \geq 0$.

[in] A

A is COMPLEX*16 array, dimension (LDA,N)

The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

[in] LDA

LDA is INTEGER

The leading dimension of the array A. $LDA \geq \max(1, M)$.

[out] B

B is COMPLEX*16 array, dimension (LDB,N)

On exit, $B = A$ in the locations specified by UPLO.

[in] LDB

LDB is INTEGER

The leading dimension of the array B. $LDB \geq \max(1, M)$.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlacpy.f —

```
* =====
*      SUBROUTINE ZLACPY( UPLO, M, N, A, LDA, B, LDB )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          UPLO
*      INTEGER            LDA, LDB, M, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16         A( LDA, * ), B( LDB, * )
*
*      ..
*
*      =====
*
*      .. Local Scalars ..
*      INTEGER            I, J
*
*      ..
*      .. External Functions ..
*      LOGICAL            LSAME
*      EXTERNAL           LSAME
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC          MIN
*
*      ..
*      .. Executable Statements ..
*
*      IF( LSAME( UPLO, 'U' ) ) THEN
*        DO 20 J = 1, N
```

```

        DO 10 I = 1, MIN( J, M )
            B( I, J ) = A( I, J )
10      CONTINUE
20      CONTINUE
*
      ELSE IF( LSAME( UPLO, 'L' ) ) THEN
        DO 40 J = 1, N
          DO 30 I = J, M
            B( I, J ) = A( I, J )
30        CONTINUE
40      CONTINUE
*
      ELSE
        DO 60 J = 1, N
          DO 50 I = 1, M
            B( I, J ) = A( I, J )
50        CONTINUE
60      CONTINUE
      END IF
*
      RETURN
*
*      End of ZLACPY
*
      END

```

— LAPACK zlacpy —

```

(defun zlacpy (uplo m n a lda b ldb$)
  (declare (type (simple-array character (*)) uplo)
    (type (f2cl-lib:integer4) ldb$ lda n m)
    (type (array f2cl-lib:complex16 (*)) b a))
  (f2cl-lib:with-multi-array-data
    ((a f2cl-lib:complex16 a-%data% a-%offset%)
     (b f2cl-lib:complex16 b-%data% b-%offset%)
     (uplo character uplo-%data% uplo-%offset%))
    (prog ((i 0) (j 0))
      (declare (type (f2cl-lib:integer4) j i))
      (cond
        ((multiple-value-bind (ret-val var-0 var-1) (lsame uplo "U")
          (declare (ignore var-1)) (when var-0 (setf uplo var-0)) ret-val)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
           ((> j n) nil)
           (tagbody
             (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
               ((> i

```

```

        (min (the f2cl-lib:integer4 j)
              (the f2cl-lib:integer4 m)))
      nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data% (i j) ((1 ldb$) (1 *)))
            b-%offset%)
      (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)))
      a-%offset%))
    label10))
  label20)))
((multiple-value-bind (ret-val var-0 var-1) (lsame uplo "L")
  (declare (ignore var-1)) (when var-0 (setf uplo var-0)) ret-val)
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
     (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data% (i j) ((1 ldb$) (1 *)))
            b-%offset%)
      (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)))
      a-%offset%))
    label30))
  label40)))
(t
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     (> i m) nil)
    (tagbody
      (setf (f2cl-lib:fref b-%data% (i j) ((1 ldb$) (1 *)))
            b-%offset%)
      (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *)))
      a-%offset%))
    label50))
  label60))))
  (go end_label)
end_label
  (return (values uplo nil nil nil nil nil nil)))
))

```

zladiv LAPACK

— zladiv.input —

```

)set break resume
)sys rm -f zladiv.output
)spool zladiv.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zladiv.help —

```

=====
zladiv examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```

      COMPLEX*16      FUNCTION ZLADIV( X, Y )

      .. Scalar Arguments ..
      COMPLEX*16      X, Y
      ..

```

Purpose:
 =====

ZLADIV := X / Y , where X and Y are complex. The computation of X / Y will not overflow on an intermediary step unless the results overflows.

Arguments:
 =====

[in] X

X is COMPLEX*16

[in] Y

Y is COMPLEX*16
The complex scalars X and Y.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zladiv.f —

```
* =====
*      COMPLEX*16      FUNCTION ZLADIV( X, Y )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,      --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      COMPLEX*16      X, Y
*      ..
*
*      =====
*
*      .. Local Scalars ..
*      DOUBLE PRECISION      ZI, ZR
*      ..
*      .. External Subroutines ..
*      EXTERNAL      DLADIV
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC      DBLE, DCMPLX, DIMAG
*      ..
*      .. Executable Statements ..
*
*      CALL DLADIV( DBLE( X ), DIMAG( X ), DBLE( Y ), DIMAG( Y ), ZR,
*      $              ZI )
*      ZLADIV = DCMPLX( ZR, ZI )
```

```

*
      RETURN
*
*      End of ZLADIV
*
      END

```

— LAPACK zladiv —

```

(defun zladiv (x y) (declare (type (f2cl-lib:complex16) y x))
  (prog
    ((zi 0.0d0) (zr 0.0d0) (zladiv #C(0.0d0 0.0d0)) (dble$ 0.0) (dimag$ 0.0))
    (declare (type (double-float) zr zi) (type (f2cl-lib:complex16) zladiv)
      (type (single-float) dimag$ dble$))
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5)
      (dladiv (f2cl-lib:dble x) (f2cl-lib:dimag x) (f2cl-lib:dble y)
        (f2cl-lib:dimag y) zr zi)
      (declare (ignore var-0 var-1 var-2 var-3)) (when var-4 (setf zr var-4))
      (when var-5 (setf zi var-5)))
    (setf zladiv (f2cl-lib:dcmplx zr zi)) (go end_label) end_label
    (return (values zladiv nil nil))))

```

zlahqr LAPACK

— zlahqr.input —

```

)set break resume
)sys rm -f zlahqr.output
)spool zlahqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlahqr.help —

```
=====
zlahqr examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====

SUBROUTINE ZLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
                  IHIZ, Z, LDZ, INFO )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, N
LOGICAL          WANTT, WANTZ
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       H( LDH, * ), W( * ), Z( LDZ, * )
```

```
..
```

Purpose:

```
=====
```

ZLAHQR is an auxiliary routine called by CHSEQR to update the eigenvalues and Schur decomposition already computed by CHSEQR, by dealing with the Hessenberg submatrix in rows and columns ILO to IHI.

Arguments:

```
=====
```

[in] WANTT

WANTT is LOGICAL

= .TRUE. : the full Schur form T is required;

= .FALSE.: only eigenvalues are required.

[in] WANTZ

WANTZ is LOGICAL

= .TRUE. : the matrix of Schur vectors Z is required;

= .FALSE.: Schur vectors are not required.

[in] N

N is INTEGER
The order of the matrix H. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER
It is assumed that H is already upper triangular in rows and columns IHI+1:N, and that $H(ILO, ILO-1) = 0$ (unless $ILO = 1$). ZLAHQQR works primarily with the Hessenberg submatrix in rows and columns ILO to IHI, but applies transformations to all of H if WANTT is .TRUE..
 $1 \leq ILO \leq \max(1, IHI)$; $IHI \leq N$.

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)
On entry, the upper Hessenberg matrix H.
On exit, if INFO is zero and if WANTT is .TRUE., then H is upper triangular in rows and columns ILO:IHI. If INFO is zero and if WANTT is .FALSE., then the contents of H are unspecified on exit. The output state of H in case INF is positive is below under the description of INFO.

[in] LDH

LDH is INTEGER
The leading dimension of the array H. $LDH \geq \max(1, N)$.

[out] W

W is COMPLEX*16 array, dimension (N)
The computed eigenvalues ILO to IHI are stored in the corresponding elements of W. If WANTT is .TRUE., the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with $W(i) = H(i, i)$.

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER
Specify the rows of Z to which transformations must be

applied if WANTZ is .TRUE..
 1 <= ILOZ <= ILO; IHI <= IHIZ <= N.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,N)
 If WANTZ is .TRUE., on entry Z must contain the current matrix Z of transformations accumulated by CHSEQR, and on exit Z has been updated; transformations are applied only to the submatrix Z(ILOZ:IHIZ,ILO:IHI).
 If WANTZ is .FALSE., Z is not referenced.

[in] LDZ

LDZ is INTEGER
 The leading dimension of the array Z. LDZ >= max(1,N).

[out] INFO

INFO is INTEGER
 = 0: successful exit
 .GT. 0: if INFO = i, ZLAHQR failed to compute all the eigenvalues ILO to IHI in a total of 30 iterations per eigenvalue; elements i+1:ihi of W contain those eigenvalues which have been successfully computed.

If INFO .GT. 0 and WANTT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit
 (*) (initial value of H)*U = U*(final value of H)
 where U is an orthogonal matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit
 (final value of Z) = (initial value of Z)*U
 where U is the orthogonal matrix in (*)
 (regardless of the value of WANTT.)

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver

NAG Ltd.

November 2011

Contributors:

=====

02-96 Based on modifications by
David Day, Sandia National Laboratory, USA

12-04 Further modifications by
Ralph Byers, University of Kansas, USA
This is a modified version of ZLAHQR from LAPACK version 3.0.
It is (1) more robust against overflow and underflow and
(2) adopts the more conservative Ahues & Tisseur stopping
criterion (LAWN 122, 1997).

— zlahqr.f —

```
* =====
*      SUBROUTINE ZLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
*      $                  IHIZ, Z, LDZ, INFO )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, N
*      LOGICAL          WANTT, WANTZ
*      ..
*      .. Array Arguments ..
*      COMPLEX*16        H( LDH, * ), W( * ), Z( LDZ, * )
*      ..
*
* =====
*
*      .. Parameters ..
*      INTEGER          ITMAX
*      PARAMETER        ( ITMAX = 30 )
*      COMPLEX*16        ZERO, ONE
*      PARAMETER        ( ZERO = ( 0.0d0, 0.0d0 ),
*      $                  ONE = ( 1.0d0, 0.0d0 ) )
*      DOUBLE PRECISION  RZERO, RONE, HALF
*      PARAMETER        ( RZERO = 0.0d0, RONE = 1.0d0, HALF = 0.5d0 )
```

```

      DOUBLE PRECISION  DAT1
      PARAMETER          ( DAT1 = 3.0d0 / 4.0d0 )
*
*   ..
*   .. Local Scalars ..
      COMPLEX*16          CDUM, H11, H11S, H22, SC, SUM, T, T1, TEMP, U,
$      V2, X, Y
      DOUBLE PRECISION    AA, AB, BA, BB, H10, H21, RTEMP, S, SAFMAX,
$      SAFMIN, SMLNUM, SX, T2, TST, ULP
      INTEGER             I, I1, I2, ITS, J, JHI, JLO, K, L, M, NH, NZ
*
*   ..
*   .. Local Arrays ..
      COMPLEX*16          V( 2 )
*
*   ..
*   .. External Functions ..
      COMPLEX*16          ZLADIV
      DOUBLE PRECISION    DLAMCH
      EXTERNAL            ZLADIV, DLAMCH
*
*   ..
*   .. External Subroutines ..
      EXTERNAL            DLABAD, ZCOPY, ZLARFG, ZSCAL
*
*   ..
*   .. Statement Functions ..
      DOUBLE PRECISION    CABS1
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC            ABS, DBLE, DCONJG, DIMAG, MAX, MIN, SQRT
*
*   ..
*   .. Statement Function definitions ..
      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
*   ..
*   .. Executable Statements ..
*
      INFO = 0
*
*   Quick return if possible
*
      IF( N.EQ.0 )
$      RETURN
      IF( ILO.EQ.IHI ) THEN
          W( ILO ) = H( ILO, ILO )
          RETURN
      END IF
*
*
*   ==== clear out the trash ====
      DO 10 J = ILO, IHI - 3
          H( J+2, J ) = ZERO
          H( J+3, J ) = ZERO
10  CONTINUE
      IF( ILO.LE.IHI-2 )
$      H( IHI, IHI-2 ) = ZERO

```

```

*      ==== ensure that subdiagonal entries are real ====
      IF( WANTT ) THEN
        JLO = 1
        JHI = N
      ELSE
        JLO = ILO
        JHI = IHI
      END IF
      DO 20 I = ILO + 1, IHI
        IF( DIMAG( H( I, I-1 ) ).NE.RZERO ) THEN
          *      ==== The following redundant normalization
          *      .      avoids problems with both gradual and
          *      .      sudden underflow in ABS(H(I,I-1)) ====
          SC = H( I, I-1 ) / CABS1( H( I, I-1 ) )
          SC = DCONJG( SC ) / ABS( SC )
          H( I, I-1 ) = ABS( H( I, I-1 ) )
          CALL ZSCAL( JHI-I+1, SC, H( I, I ), LDH )
          CALL ZSCAL( MIN( JHI, I+1 )-JLO+1, DCONJG( SC ),
$           H( JLO, I ), 1 )
          IF( WANTZ )
$           CALL ZSCAL( IHI-ILOZ+1, DCONJG( SC ), Z( ILOZ, I ), 1 )
          END IF
        20 CONTINUE
      *
      NH = IHI - ILO + 1
      NZ = IHI - ILOZ + 1
      *
      *      Set machine-dependent constants for the stopping criterion.
      *
      SAFMIN = DLAMCH( 'SAFE MINIMUM' )
      SAFMAX = RONE / SAFMIN
      CALL DLABAD( SAFMIN, SAFMAX )
      ULP = DLAMCH( 'PRECISION' )
      SMLNUM = SAFMIN*( DBLE( NH ) / ULP )
      *
      *      I1 and I2 are the indices of the first row and last column of H
      *      to which transformations must be applied. If eigenvalues only are
      *      being computed, I1 and I2 are set inside the main loop.
      *
      IF( WANTT ) THEN
        I1 = 1
        I2 = N
      END IF
      *
      *      The main loop begins here. I is the loop index and decreases from
      *      IHI to ILO in steps of 1. Each iteration of the loop works
      *      with the active submatrix in rows and columns L to I.
      *      Eigenvalues I+1 to IHI have already converged. Either L = ILO, or
      *      H(L,L-1) is negligible so that the matrix splits.
      *

```



```

      I = IHI
30  CONTINUE
      IF( I.LT.ILO )
        $  GO TO 150
*
*    Perform QR iterations on rows and columns ILO to I until a
*    submatrix of order 1 splits off at the bottom because a
*    subdiagonal element has become negligible.
*
      L = ILO
      DO 130 ITS = 0, ITMAX
*
*    Look for a single small subdiagonal element.
*
      DO 40 K = I, L + 1, -1
        IF( CABS1( H( K, K-1 ) ) .LE. SMLNUM )
          $  GO TO 50
        TST = CABS1( H( K-1, K-1 ) ) + CABS1( H( K, K ) )
        IF( TST.EQ.ZERO ) THEN
          IF( K-2.GE.ILO )
            $  TST = TST + ABS( DBLE( H( K-1, K-2 ) ) )
          IF( K+1.LE.IHI )
            $  TST = TST + ABS( DBLE( H( K+1, K ) ) )
        END IF
        ==== The following is a conservative small subdiagonal
        .   deflation criterion due to Ahues & Tisseur (LAWN 122,
        .   1997). It has better mathematical foundation and
        .   improves accuracy in some examples. ====
        IF( ABS( DBLE( H( K, K-1 ) ) ) .LE. ULP*TST ) THEN
          AB = MAX( CABS1( H( K, K-1 ) ), CABS1( H( K-1, K ) ) )
          BA = MIN( CABS1( H( K, K-1 ) ), CABS1( H( K-1, K ) ) )
          AA = MAX( CABS1( H( K, K ) ),
            $  CABS1( H( K-1, K-1 ) - H( K, K ) ) )
          BB = MIN( CABS1( H( K, K ) ),
            $  CABS1( H( K-1, K-1 ) - H( K, K ) ) )
          S = AA + AB
          IF( BA*( AB / S ) .LE. MAX( SMLNUM,
            $  ULP*( BB*( AA / S ) ) ) ) GO TO 50
        END IF
40  CONTINUE
50  CONTINUE
      L = K
      IF( L.GT.ILO ) THEN
*
*    H(L,L-1) is negligible
*
        H( L, L-1 ) = ZERO
      END IF
*
*    Exit from loop if a submatrix of order 1 has split off.

```

```

*
*      IF( L.GE.I )
*      $      GO TO 140
*
*      Now the active submatrix is in rows and columns L to I. If
*      eigenvalues only are being computed, only the active submatrix
*      need be transformed.
*
*      IF( .NOT.WANTT ) THEN
*          I1 = L
*          I2 = I
*      END IF
*
*      IF( ITS.EQ.10 ) THEN
*
*          Exceptional shift.
*
*          S = DAT1*ABS( DBLE( H( L+1, L ) ) )
*          T = S + H( L, L )
*      ELSE IF( ITS.EQ.20 ) THEN
*
*          Exceptional shift.
*
*          S = DAT1*ABS( DBLE( H( I, I-1 ) ) )
*          T = S + H( I, I )
*      ELSE
*
*          Wilkinson's shift.
*
*          T = H( I, I )
*          U = SQRT( H( I-1, I ) )*SQRT( H( I, I-1 ) )
*          S = CABS1( U )
*          IF( S.NE.RZERO ) THEN
*              X = HALF*( H( I-1, I-1 )-T )
*              SX = CABS1( X )
*              S = MAX( S, CABS1( X ) )
*              Y = S*SQRT( ( X / S )**2+( U / S )**2 )
*              IF( SX.GT.RZERO ) THEN
*                  IF( DBLE( X / SX )*DBLE( Y )+DIMAG( X / SX )*
*                      DIMAG( Y ).LT.RZERO ) Y = -Y
*                  $
*              END IF
*              T = T - U*ZLADIV( U, ( X+Y ) )
*          END IF
*      END IF
*
*      Look for two consecutive small subdiagonal elements.
*
*      DO 60 M = I - 1, L + 1, -1
*
*          Determine the effect of starting the single-shift QR

```

```

*          iteration at row M, and see if this would make H(M,M-1)
*          negligible.
*
      H11 = H( M, M )
      H22 = H( M+1, M+1 )
      H11S = H11 - T
      H21 = DBLE( H( M+1, M ) )
      S = CABS1( H11S ) + ABS( H21 )
      H11S = H11S / S
      H21 = H21 / S
      V( 1 ) = H11S
      V( 2 ) = H21
      H10 = DBLE( H( M, M-1 ) )
      IF( ABS( H10 )*ABS( H21 ).LE.ULP*
$          ( CABS1( H11S )*( CABS1( H11 )+CABS1( H22 ) ) ) )
$          GO TO 70
60    CONTINUE
      H11 = H( L, L )
      H22 = H( L+1, L+1 )
      H11S = H11 - T
      H21 = DBLE( H( L+1, L ) )
      S = CABS1( H11S ) + ABS( H21 )
      H11S = H11S / S
      H21 = H21 / S
      V( 1 ) = H11S
      V( 2 ) = H21
70    CONTINUE
*
*          Single-shift QR step
*
      DO 120 K = M, I - 1
*
*          The first iteration of this loop determines a reflection G
*          from the vector V and applies it from left and right to H,
*          thus creating a nonzero bulge below the subdiagonal.
*
*          Each subsequent iteration determines a reflection G to
*          restore the Hessenberg form in the (K-1)th column, and thus
*          chases the bulge one step toward the bottom of the active
*          submatrix.
*
*          V(2) is always real before the call to ZLARFG, and hence
*          after the call T2 ( = T1*V(2) ) is also real.
*
      IF( K.GT.M )
$          CALL ZCOPY( 2, H( K, K-1 ), 1, V, 1 )
      CALL ZLARFG( 2, V( 1 ), V( 2 ), 1, T1 )
      IF( K.GT.M ) THEN
          H( K, K-1 ) = V( 1 )
          H( K+1, K-1 ) = ZERO

```

```

      END IF
      V2 = V( 2 )
      T2 = DBLE( T1*V2 )
*
*      Apply G from the left to transform the rows of the matrix
*      in columns K to I2.
*
      DO 80 J = K, I2
        SUM = DCONJG( T1 )*H( K, J ) + T2*H( K+1, J )
        H( K, J ) = H( K, J ) - SUM
        H( K+1, J ) = H( K+1, J ) - SUM*V2
80      CONTINUE
*
*      Apply G from the right to transform the columns of the
*      matrix in rows I1 to min(K+2,I).
*
      DO 90 J = I1, MIN( K+2, I )
        SUM = T1*H( J, K ) + T2*H( J, K+1 )
        H( J, K ) = H( J, K ) - SUM
        H( J, K+1 ) = H( J, K+1 ) - SUM*DCONJG( V2 )
90      CONTINUE
*
      IF( WANTZ ) THEN
*
*        Accumulate transformations in the matrix Z
*
        DO 100 J = ILOZ, IHIZ
          SUM = T1*Z( J, K ) + T2*Z( J, K+1 )
          Z( J, K ) = Z( J, K ) - SUM
          Z( J, K+1 ) = Z( J, K+1 ) - SUM*DCONJG( V2 )
100        CONTINUE
      END IF
*
      IF( K.EQ.M .AND. M.GT.L ) THEN
*
*        If the QR step was started at row M > L because two
*        consecutive small subdiagonals were found, then extra
*        scaling must be performed to ensure that H(M,M-1) remains
*        real.
*
        TEMP = ONE - T1
        TEMP = TEMP / ABS( TEMP )
        H( M+1, M ) = H( M+1, M )*DCONJG( TEMP )
        IF( M+2.LE.I )
          H( M+2, M+1 ) = H( M+2, M+1 )*TEMP
        DO 110 J = M, I
          IF( J.NE.M+1 ) THEN
            IF( I2.GT.J )
              CALL ZSCAL( I2-J, TEMP, H( J, J+1 ), LDH )
              CALL ZSCAL( J-I1, DCONJG( TEMP ), H( I1, J ), 1 )

```

```

                IF( WANTZ ) THEN
                    CALL ZSCAL( NZ, DCONJG( TEMP ), Z( ILOZ, J ),
$                      1 )
                END IF
            END IF
110        CONTINUE
        END IF
120    CONTINUE
*
*        Ensure that H(I,I-1) is real.
*
        TEMP = H( I, I-1 )
        IF( DIMAG( TEMP ).NE.RZERO ) THEN
            RTEMP = ABS( TEMP )
            H( I, I-1 ) = RTEMP
            TEMP = TEMP / RTEMP
            IF( I2.GT.I )
$                CALL ZSCAL( I2-I, DCONJG( TEMP ), H( I, I+1 ), LDH )
            CALL ZSCAL( I-I1, TEMP, H( I1, I ), 1 )
            IF( WANTZ ) THEN
                CALL ZSCAL( NZ, TEMP, Z( ILOZ, I ), 1 )
            END IF
        END IF
*
130 CONTINUE
*
*        Failure to converge in remaining number of iterations
*
        INFO = I
        RETURN
*
140 CONTINUE
*
*        H(I,I-1) is negligible: one eigenvalue has converged.
*
        W( I ) = H( I, I )
*
*        return to start of the main loop with new value of I.
*
        I = L - 1
        GO TO 30
*
150 CONTINUE
        RETURN
*
*        End of ZLAHQRL
*
        END

```

— LAPACK zlahqr —

```

(let*
  ((itmax 30) (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (rzero 0.0d0)
   (rone 1.0d0) (half 0.5d0) (dat1 (f2cl-lib:f2cl/ 3.0d0 4.0d0)))
  (declare (type (f2cl-lib:integer4 30 30) itmax)
            (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
            (type (double-float 0.0d0 0.0d0) rzero)
            (type (double-float 1.0d0 1.0d0) rone) (type (double-float 0.5d0 0.5d0) half)
            (type (double-float) dat1) (ignorable itmax zero one rzero rone half dat1))
  (defun zlahqr (wantt wantz n ilo ihi h ldh w iloz ihiz z ldz info)
    (declare (type f2cl-lib:logical wantz wantt)
              (type (f2cl-lib:integer4) info ldz ihiz iloz ldh ihi ilo n)
              (type (array f2cl-lib:complex16 (*)) z w h))
    (f2cl-lib:with-multi-array-data
      ((h f2cl-lib:complex16 h-%data% h-%offset%)
       (w f2cl-lib:complex16 w-%data% w-%offset%)
       (z f2cl-lib:complex16 z-%data% z-%offset%))
      (labels
        ((cabs1 (cdum)
          (+ (abs (f2cl-lib:dbler cdum)) (abs (f2cl-lib:dimag cdum)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                           (values double-float &rest t)) cabs1))
          (prog
            ((v (make-array 2 :element-type 'f2cl-lib:complex16)) (i 0) (i1 0)
             (i2 0)
             (its 0) (j 0) (jhi 0) (jlo 0) (k 0) (l 0) (m 0) (nh 0) (nz 0)
             (aa 0.0d0)
             (ab 0.0d0) (ba 0.0d0) (bb 0.0d0) (h10 0.0d0) (h21 0.0d0)
             (rtemp 0.0d0)
             (s 0.0d0) (safmax 0.0d0) (safmin 0.0d0) (smlnum 0.0d0)
             (sx 0.0d0) (t2 0.0d0)
             (tst 0.0d0) (ulp 0.0d0) (cdum #C(0.0d0 0.0d0)) (h11 #C(0.0d0 0.0d0))
             (h11s #C(0.0d0 0.0d0)) (h22 #C(0.0d0 0.0d0)) (sc #C(0.0d0 0.0d0))
             (sum #C(0.0d0 0.0d0)) (t$ #C(0.0d0 0.0d0)) (t1 #C(0.0d0 0.0d0))
             (temp #C(0.0d0 0.0d0)) (u #C(0.0d0 0.0d0)) (v2 #C(0.0d0 0.0d0))
             (x #C(0.0d0 0.0d0)) (y #C(0.0d0 0.0d0)) (dconjg$ 0.0))
            (declare (type (array f2cl-lib:complex16 (2)) v)
                      (type (f2cl-lib:integer4) nz nh m l k jlo jhi j its i2 i1 i)
                      (type (double-float)
                        ulp tst t2 sx smlnum safmin safmax s rtemp h21 h10 bb
                        ba ab aa)
                      (type (f2cl-lib:complex16)
                        y x v2 u temp t1 t$ sum sc h22 h11s h11 cdum)
                      (type (single-float) dconjg$))

```

```

(setf info 0) (if (= n 0) (go end_label))
(cond
  ((= ilo ihi)
    (setf (f2cl-lib:fref w-%data% (ilo) ((1 *)) w-%offset%)
      (f2cl-lib:fref h-%data% (ilo ilo) ((1 ldh) (1 *)) h-%offset%))
    (go end_label)))
(f2cl-lib:fdo (j ilo (f2cl-lib:int-add j 1))
  (> j
    (f2cl-lib:int-add ihi (f2cl-lib:int-sub 3)))
    nil)
  (tagbody
    (setf
      (f2cl-lib:fref h-%data% ((f2cl-lib:int-add j 2) j)
        ((1 ldh) (1 *))
        h-%offset%)
      zero)
    (setf
      (f2cl-lib:fref h-%data% ((f2cl-lib:int-add j 3) j)
        ((1 ldh) (1 *))
        h-%offset%)
      zero)
    label10))
(if (<= ilo (f2cl-lib:int-sub ihi 2))
  (setf
    (f2cl-lib:fref h-%data%
      (ihi (f2cl-lib:int-sub ihi 2)) ((1 ldh) (1 *))
      h-%offset%)
    zero))
(cond (wantt (setf jlo 1) (setf jhi n))
  (t (setf jlo ilo) (setf jhi ihi)))
(f2cl-lib:fdo (i (f2cl-lib:int-add ilo 1) (f2cl-lib:int-add i 1))
  (> i ihi)
  nil)
  (tagbody
    (cond
      ((/=
        (f2cl-lib:dimag
          (f2cl-lib:fref h (i (f2cl-lib:int-add i
            (f2cl-lib:int-sub 1))))
            ((1 ldh) (1 *))))
        rzero)
      (setf sc
        (/
          (f2cl-lib:fref h-%data% (i (f2cl-lib:int-sub i 1))
            ((1 ldh) (1 *))
            h-%offset%)
          (cabs1
            (f2cl-lib:fref h-%data% (i (f2cl-lib:int-sub i 1))
              ((1 ldh) (1 *))
              h-%offset%))))))

```

```

(setf sc (coerce (/ (f2cl-lib:dconjg sc) (abs sc))
                  'f2cl-lib:complex16))
(setf
  (f2cl-lib:fref h-%data% (i (f2cl-lib:int-sub i 1))
    ((1 ldh) (1 *))
    h-%offset%)
  (coerce
    (abs
      (f2cl-lib:fref h-%data% (i (f2cl-lib:int-sub i 1))
        ((1 ldh) (1 *))
        h-%offset%))
    'f2cl-lib:complex16))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal (f2cl-lib:int-add (f2cl-lib:int-sub jhi i) 1) sc
    (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
      (i i) ((1 ldh) (1 *))
      h-%offset%)
    ldh)
  (declare (ignore var-0 var-2))
  (when var-1 (setf sc var-1))
  (when var-3 (setf ldh var-3)))
(zscal
  (f2cl-lib:int-add
    (f2cl-lib:int-sub
      (min (the f2cl-lib:integer4 jhi)
        (the f2cl-lib:integer4 (f2cl-lib:int-add i 1)))
      jlo)
    1)
  (f2cl-lib:dconjg sc)
  (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
    (jlo i) ((1 ldh) (1 *))
    h-%offset%)
  1)
(if wantz
  (zscal (f2cl-lib:int-add (f2cl-lib:int-sub ihiz iloz) 1)
    (f2cl-lib:dconjg sc)
    (f2cl-lib:array-slice z-%data% f2cl-lib:complex16
      (ilo z i)
      ((1 ldz) (1 *)) z-%offset%)
    1))))
label20))
(setf nh (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))
(setf nz (f2cl-lib:int-add (f2cl-lib:int-sub ihiz iloz) 1))
(setf safmin (dlamch "SAFE MINIMUM")) (setf safmax (/ rone safmin))
(multiple-value-bind (var-0 var-1) (dlabad safmin safmax)
  (declare (ignore))
  (when var-0 (setf safmin var-0)) (when var-1 (setf safmax var-1)))
(setf ulp (dlamch "PRECISION"))
(setf smlnum (* safmin (/ (f2cl-lib:double nh) ulp)))
(cond (wantt (setf i1 1) (setf i2 n))) (setf i ihi) label30

```



```

(if (< i ilo) (go label150)) (setf l ilo)
(f2cl-lib:fdo (its 0 (f2cl-lib:int-add its 1))
  (> its itmax) nil)
  (tagbody
    (f2cl-lib:fdo (k i (f2cl-lib:int-add k
      (f2cl-lib:int-sub 1)))
      (> k
        (f2cl-lib:int-add 1 1))
        nil)
    (tagbody
      (if
        (<=
          (cabs1
            (f2cl-lib:fref h-%data% (k (f2cl-lib:int-sub k 1))
              ((1 ldh) (1 *))
              h-%offset%))
            smlnum)
          (go label150))
      (setf tst
        (+
          (cabs1
            (f2cl-lib:fref h-%data% ((f2cl-lib:int-sub k 1)
              (f2cl-lib:int-sub k 1))
              ((1 ldh) (1 *)) h-%offset%))
            (cabs1 (f2cl-lib:fref h-%data% (k k) ((1 ldh) (1 *))
              h-%offset%))))))
      (cond
        ((= tst zero)
          (if (>= (f2cl-lib:int-sub k 2) ilo)
            (setf tst
              (+ tst
                (abs
                  (f2cl-lib:double
                    (f2cl-lib:fref h-%data%
                      ((f2cl-lib:int-sub k 1)
                        (f2cl-lib:int-sub k 2)) ((1 ldh) (1 *))
                      h-%offset%)))))))
            (if (<= (f2cl-lib:int-add k 1) ihi)
              (setf tst
                (+ tst
                  (abs
                    (f2cl-lib:double
                      (f2cl-lib:fref h-%data%
                        ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
                        h-%offset%))))))))))
        (t
          (cond
            ((<=
              (abs
                (f2cl-lib:double
                  (f2cl-lib:fref h

```

```

      (k (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
      ((1 ldh) (1 *))))
    (* ulp tst))
  (setf ab
    (max
      (cabs1
        (f2cl-lib:fref h-%data% (k (f2cl-lib:int-sub k 1))
          ((1 ldh) (1 *))
          h-%offset%))
      (cabs1
        (f2cl-lib:fref h-%data% ((f2cl-lib:int-sub k 1) k)
          ((1 ldh) (1 *))
          h-%offset%))))
  (setf ba
    (min
      (cabs1
        (f2cl-lib:fref h-%data% (k (f2cl-lib:int-sub k 1))
          ((1 ldh) (1 *))
          h-%offset%))
      (cabs1
        (f2cl-lib:fref h-%data% ((f2cl-lib:int-sub k 1) k)
          ((1 ldh) (1 *))
          h-%offset%))))
  (setf aa
    (max (cabs1 (f2cl-lib:fref h-%data% (k k) ((1 ldh)
      (1 *)) h-%offset%))
      (cabs1
        (-
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-sub k 1) (f2cl-lib:int-sub k 1))
            ((1 ldh) (1 *))
            h-%offset%)
          (f2cl-lib:fref h-%data% (k k) ((1 ldh) (1 *))
            h-%offset%))))))
  (setf bb
    (min (cabs1 (f2cl-lib:fref h-%data% (k k) ((1 ldh)
      (1 *)) h-%offset%))
      (cabs1
        (-
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-sub k 1) (f2cl-lib:int-sub k 1))
            ((1 ldh) (1 *))
            h-%offset%)
          (f2cl-lib:fref h-%data% (k k) ((1 ldh) (1 *))
            h-%offset%))))))
  (setf s (+ aa ab))
  (if (<= (* ba (/ ab s))
    (max smlnum (* ulp (* bb (/ aa s))))
    (go label150)))
label140))

```

```

label150 (setf 1 k)
(cond
  (> 1 ilo)
  (setf
    (f2cl-lib:fref h-%data% (1 (f2cl-lib:int-sub 1 1))
      ((1 ldh) (1 *))
      h-%offset%)
    zero)))
(if (>= 1 i) (go label140))
(cond ((not wantt) (setf i1 1) (setf i2 i)))
(cond
  (= its 10)
  (setf s
    (* dat1
      (abs
        (f2cl-lib:dbple
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add 1 1) 1) ((1 ldh) (1 *))
            h-%offset%))))))
  (setf t$ (+ s (f2cl-lib:fref h-%data% (1 1) ((1 ldh)
    (1 *)) h-%offset%))))
(= its 20)
(setf s
  (* dat1
    (abs
      (f2cl-lib:dbple
        (f2cl-lib:fref h-%data%
          (i (f2cl-lib:int-sub i 1)) ((1 ldh) (1 *))
          h-%offset%))))))
  (setf t$ (+ s (f2cl-lib:fref h-%data%
    (i i) ((1 ldh) (1 *)) h-%offset%))))
(t (setf t$ (f2cl-lib:fref h-%data%
  (i i) ((1 ldh) (1 *)) h-%offset%))
  (setf u
    (*
      (f2cl-lib:fsqrt
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-sub i 1) i) ((1 ldh) (1 *))
          h-%offset%))
      (f2cl-lib:fsqrt
        (f2cl-lib:fref h-%data%
          (i (f2cl-lib:int-sub i 1)) ((1 ldh) (1 *))
          h-%offset%))))
    (setf s (cabs1 u))
  (cond
    (/= s rzero)
    (setf x
      (* half
        (-
          (f2cl-lib:fref h-%data%

```

```

      ((f2cl-lib:int-sub i 1) (f2cl-lib:int-sub i 1))
      ((1 ldh) (1 *))
      h-%offset%)
      t$)))
    (setf sx (cabs1 x)) (setf s (max s (cabs1 x)))
    (setf y (* s (f2cl-lib:fsqrt (+ (expt (/ x s) 2)
                                     (expt (/ u s) 2)))))

    (cond
      ((> sx rzero)
        (if
          (<
            (+ (* (f2cl-lib:db1e (/ x sx)) (f2cl-lib:db1e y))
              (* (f2cl-lib:dimag (/ x sx)) (f2cl-lib:dimag y)))
            rzero)
          (setf y (- y))))
        (setf t$ (- t$ (* u (zladd u (+ x y)))))))
    (f2cl-lib:fdo (m (f2cl-lib:int-add i (f2cl-lib:int-sub 1))
                  (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
      ((> m (f2cl-lib:int-add l 1))
        nil)
      (tagbody
        (setf h11
          (f2cl-lib:fref h-%data% (m m) ((1 ldh) (1 *))
            h-%offset%))
        (setf h22
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add m 1) (f2cl-lib:int-add m 1))
            ((1 ldh) (1 *)) h-%offset%))
        (setf h11s (- h11 t$))
        (setf h21
          (f2cl-lib:db1e
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-add m 1) m) ((1 ldh) (1 *))
              h-%offset%)))
        (setf s (+ (cabs1 h11s) (abs h21)))
        (setf h11s (/ h11s s))
        (setf h21 (/ h21 s))
        (setf (f2cl-lib:fref v (1) ((1 2))) h11s)
        (setf (f2cl-lib:fref v (2) ((1 2)))
          (coerce h21 'f2cl-lib:complex16))
        (setf h10
          (f2cl-lib:db1e
            (f2cl-lib:fref h-%data%
              (m (f2cl-lib:int-sub m 1)) ((1 ldh) (1 *))
              h-%offset%)))
        (if
          (<= (* (abs h10) (abs h21))
            (* ulp (* (cabs1 h11s) (+ (cabs1 h11) (cabs1 h22)))))
          (go label170))
        label160))

```

```

(setf h11
  (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *)) h-%offset%))
(setf h22
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add 1 1) (f2cl-lib:int-add 1 1))
    ((1 ldh) (1 *)) h-%offset%))
(setf h11s (- h11 t$))
(setf h21
  (f2cl-lib:double
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add 1 1) 1) ((1 ldh) (1 *))
      h-%offset%)))
(setf s (+ (cabs1 h11s) (abs h21))) (setf h11s (/ h11s s))
(setf h21 (/ h21 s))
(setf (f2cl-lib:fref v (1) ((1 2))) h11s)
(setf (f2cl-lib:fref v (2) ((1 2)))
  (coerce h21 'f2cl-lib:complex16))
label70
(f2cl-lib:fdo (k m (f2cl-lib:int-add k 1))
  (> k
    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    nil)
(tagbody
  (if (> k m)
    (zcopy 2
      (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
        (k (f2cl-lib:int-sub k 1)) ((1 ldh) (1 *)) h-%offset%)
        1 v 1))
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
      (zlarfg 2 (f2cl-lib:array-slice v f2cl-lib:complex16
        (1) ((1 2)))
        (f2cl-lib:array-slice v f2cl-lib:complex16
          (2) ((1 2))) 1 t1)
      (declare (ignore var-0 var-1 var-2 var-3))
      (when var-4 (setf t1 var-4)))
    (cond
      ((> k m)
        (setf
          (f2cl-lib:fref h-%data%
            (k (f2cl-lib:int-sub k 1)) ((1 ldh) (1 *))
            h-%offset%)
          (f2cl-lib:fref v (1) ((1 2))))
        (setf
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 1) (f2cl-lib:int-sub k 1))
            ((1 ldh) (1 *)) h-%offset%)
          zero)))
      (setf v2 (f2cl-lib:fref v (2) ((1 2))))
      (setf t2 (f2cl-lib:double (* t1 v2)))
      (f2cl-lib:fdo (j k (f2cl-lib:int-add j 1))

```

```

(< j i2) nil)
  (tagbody
    (setf sum
      (+
        (* (f2cl-lib:dconjg t1)
          (f2cl-lib:fref h-%data% (k j)
            ((1 ldh) (1 *)) h-%offset%))
        (* t2
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
            h-%offset%))))
    (setf (f2cl-lib:fref h-%data% (k j)
      ((1 ldh) (1 *)) h-%offset%)
      (- (f2cl-lib:fref h-%data% (k j)
        ((1 ldh) (1 *)) h-%offset%) sum))
    (setf
      (f2cl-lib:fref h-%data%
        ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
        h-%offset%)
      (-
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
          h-%offset%)
        (* sum v2)))
    label80))
(f2cl-lib:fdo (j i1 (f2cl-lib:int-add j 1))
  (< j
    (min (the f2cl-lib:integer4 (f2cl-lib:int-add k 2))
      (the f2cl-lib:integer4 i)))
    nil)
  (tagbody
    (setf sum
      (+ (* t1 (f2cl-lib:fref h-%data%
        (j k) ((1 ldh) (1 *)) h-%offset%))
        (* t2
          (f2cl-lib:fref h-%data%
            (j (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
            h-%offset%))))
    (setf (f2cl-lib:fref h-%data%
      (j k) ((1 ldh) (1 *)) h-%offset%)
      (- (f2cl-lib:fref h-%data% (j k)
        ((1 ldh) (1 *)) h-%offset%) sum))
    (setf
      (f2cl-lib:fref h-%data%
        (j (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
        h-%offset%)
      (-
        (f2cl-lib:fref h-%data%
          (j (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
          h-%offset%)

```

```

        (* sum (f2cl-lib:dconjg v2))))
    label190))
(cond
  (wantz
    (f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
      (> j ihiz) nil)
    (tagbody
      (setf sum
        (+ (* t1 (f2cl-lib:fref z-%data%
          (j k) ((1 ldz) (1 *)) z-%offset%))
          (* t2
            (f2cl-lib:fref z-%data%
              (j (f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
              z-%offset%))))
        (setf (f2cl-lib:fref z-%data%
          (j k) ((1 ldz) (1 *)) z-%offset%)
          (- (f2cl-lib:fref z-%data%
            (j k) ((1 ldz) (1 *)) z-%offset%) sum))
        (setf
          (f2cl-lib:fref z-%data%
            (j (f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
            z-%offset%)
          (-
            (f2cl-lib:fref z-%data%
              (j (f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
              z-%offset%)
            (* sum (f2cl-lib:dconjg v2))))
          label100))))
  (cond
    ((and (= k m) (> m 1)) (setf temp (- one t1))
      (setf temp (/ temp (abs temp)))
      (setf
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add m 1) m) ((1 ldh) (1 *))
          h-%offset%)
        (coerce
          (*
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-add m 1) m) ((1 ldh) (1 *))
              h-%offset%)
            (f2cl-lib:dconjg temp))
          'f2cl-lib:complex16))
      (if (<= (f2cl-lib:int-add m 2) i)
        (setf
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add m 2) (f2cl-lib:int-add m 1))
            ((1 ldh) (1 *)) h-%offset%)
          (*
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-add m 2)

```

```

(f2cl-lib:int-add m 1)) ((1 ldh) (1 *))
h-%offset%)
temp)))
(f2cl-lib:fdo (j m (f2cl-lib:int-add j 1))
(> j i) nil)
(tagbody
(cond
(=/= j (f2cl-lib:int-add m 1))
(if (> i2 j)
(multiple-value-bind (var-0 var-1 var-2 var-3)
(zscal (f2cl-lib:int-sub i2 j) temp
(f2cl-lib:array-slice h-%data%
f2cl-lib:complex16
(j (f2cl-lib:int-add j 1))
((1 ldh) (1 *)) h-%offset%)
ldh)
(declare (ignore var-0 var-2))
(when var-1 (setf temp var-1))
(when var-3 (setf ldh var-3))))
(zscal (f2cl-lib:int-sub j i1)
(f2cl-lib:dconjg temp)
(f2cl-lib:array-slice h-%data%
f2cl-lib:complex16 (i1 j)
((1 ldh) (1 *)) h-%offset%)
1)
(cond
(wantz
(multiple-value-bind (var-0 var-1 var-2 var-3)
(zscal nz (f2cl-lib:dconjg temp)
(f2cl-lib:array-slice z-%data%
f2cl-lib:complex16 (iloz j)
((1 ldz) (1 *)) z-%offset%)
1)
(declare (ignore var-1 var-2 var-3))
(when var-0 (setf nz var-0))))))
label110))))
label120))
(setf temp
(f2cl-lib:fref h-%data%
(i (f2cl-lib:int-sub i 1)) ((1 ldh) (1 *))
h-%offset%))
(cond
(=/= (f2cl-lib:dimag temp) rzero) (setf rtemp (abs temp))
(setf
(f2cl-lib:fref h-%data%
(i (f2cl-lib:int-sub i 1)) ((1 ldh) (1 *))
h-%offset%)
(coerce rtemp 'f2cl-lib:complex16))
(setf temp (/ temp rtemp))
(if (> i2 i)

```



```

(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal (f2cl-lib:int-sub i2 i) (f2cl-lib:dconjg temp)
    (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
      (i (f2cl-lib:int-add i 1)) ((1 ldh) (1 *)) h-%offset%)
    ldh)
  (declare (ignore var-0 var-1 var-2))
  (when var-3 (setf ldh var-3))))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal (f2cl-lib:int-sub i i1) temp
    (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
      (i1 i) ((1 ldh) (1 *))
      h-%offset%)
    1)
  (declare (ignore var-0 var-2 var-3))
  (when var-1 (setf temp var-1)))
(cond
  (wantz
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (zscal nz temp
        (f2cl-lib:array-slice z-%data% f2cl-lib:complex16
          (iloz i)
          ((1 ldz) (1 *)) z-%offset%)
        1)
      (declare (ignore var-2 var-3))
      (when var-0 (setf nz var-0))
      (when var-1 (setf temp var-1))))))
  label130))
(setf info i) (go end_label) label140
(setf (f2cl-lib:fref w-%data% (i) ((1 *)) w-%offset%)
  (f2cl-lib:fref h-%data% (i i) ((1 ldh) (1 *)) h-%offset%))
(setf i (f2cl-lib:int-sub 1 1)) (go label30) label150 (go end_label)
end_label
(return
  (values nil nil nil nil nil nil ldh nil nil nil nil nil info))))
)))

```

zlahr2 LAPACK

— zlahr2.input —

```

)set break resume
)sys rm -f zlahr2.output
)spool zlahr2.output
)set message test on
)set message auto off

```

```
)clear all

)spool
)lisp (bye)
```

— zlahr2.help —

```
=====
zlahr2 examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLAHR2( N, K, NB, A, LDA, TAU, T, LDT, Y, LDY )
```

```
.. Scalar Arguments ..
```

```
INTEGER          K, LDA, LDT, LDY, N, NB
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       A( LDA, * ), T( LDT, NB ), TAU( NB ),
```

```
$               Y( LDY, NB )
```

```
..
```

Purpose:

```
=====
```

ZLAHR2 reduces the first NB columns of A complex general n-BY-(n-k+1) matrix A so that elements below the k-th subdiagonal are zero. The reduction is performed by a unitary similarity transformation $Q^*H * A * Q$. The routine returns the matrices V and T which determine Q as a block reflector $I - V^*T^*V^*H$, and also the matrix $Y = A * V * T$.

This is an auxiliary routine called by ZGHERD.

Arguments:

```
=====
```

[in] N

N is INTEGER
The order of the matrix A.

[in] K

K is INTEGER
The offset for the reduction. Elements below the k-th
subdiagonal in the first NB columns are reduced to zero.
 $K < N$.

[in] NB

NB is INTEGER
The number of columns to be reduced.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N-K+1)
On entry, the n-by-(n-k+1) general matrix A.
On exit, the elements on and above the k-th subdiagonal in
the first NB columns are overwritten with the corresponding
elements of the reduced matrix; the elements below the k-th
subdiagonal, with the array TAU, represent the matrix Q as a
product of elementary reflectors. The other columns of A are
unchanged. See Further Details.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

[out] TAU

TAU is COMPLEX*16 array, dimension (NB)
The scalar factors of the elementary reflectors. See Further
Details.

[out] T

T is COMPLEX*16 array, dimension (LDT,NB)
The upper triangular matrix T.

[in] LDT

LDT is INTEGER
The leading dimension of the array T. $LDT \geq NB$.

[out] Y

Y is COMPLEX*16 array, dimension (LDY,NB)
 The n-by-nb matrix Y.

[in] LDY

LDY is INTEGER
 The leading dimension of the array Y. LDY \geq N.

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:
 =====

The matrix Q is represented as a product of nb elementary reflectors

$$Q = H(1) H(2) \dots H(nb).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v^{**}H$$

where tau is a complex scalar, and v is a complex vector with
 $v(1:i+k-1) = 0$, $v(i+k) = 1$; $v(i+k+1:n)$ is stored on exit in
 $A(i+k+1:n,i)$, and tau in TAU(i).

The elements of the vectors v together form the (n-k+1)-by-nb matrix
 V which is needed, with T and Y, to apply the transformation to the
 unreduced part of the matrix, using an update of the form:
 $A := (I - V*T*V^{**}H) * (A - Y*V^{**}H).$

The contents of A on exit are illustrated by the following example
 with n = 7, k = 3 and nb = 2:

```
( a  a  a  a  a )
( a  a  a  a  a )
( a  a  a  a  a )
( h  h  a  a  a )
( v1 h  a  a  a )
( v1 v2 a  a  a )
( v1 v2 a  a  a )
```

where a denotes an element of the original matrix A, h denotes a

modified element of the upper Hessenberg matrix H, and vi denotes an element of the vector defining H(i).

This subroutine is a slight modification of LAPACK-3.0's DLAHRD incorporating improvements proposed by Quintana-Orti and Van de Geijn. Note that the entries of A(1:K,2:NB) differ from those returned by the original LAPACK-3.0's DLAHRD routine. (This subroutine is not backward compatible with LAPACK-3.0's DLAHRD.)

References:

=====

Gregorio Quintana-Orti and Robert van de Geijn, "Improving the performance of reduction to Hessenberg form," ACM Transactions on Mathematical Software, 32(2):180-194, June 2006.

— zlahr2.f —

```
* =====
*      SUBROUTINE ZLAHR2( N, K, NB, A, LDA, TAU, T, LDT, Y, LDY )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          K, LDA, LDT, LDY, N, NB
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * ), T( LDT, NB ), TAU( NB ),
*      $                Y( LDY, NB )
*      ..
*
*      =====
*
*      .. Parameters ..
*      COMPLEX*16       ZERO, ONE
*      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ),
*      $                ONE = ( 1.0D+0, 0.0D+0 ) )
*      ..
*      .. Local Scalars ..
*      INTEGER          I
*      COMPLEX*16       EI
*      ..
*      .. External Subroutines ..
```

```

EXTERNAL          ZAXPY, ZCOPY, ZGEMM, ZGEMV, ZLACPY,
$                ZLARFG, ZSCAL, ZTRMM, ZTRMV, ZLACGV
*
* ..
* .. Intrinsic Functions ..
INTRINSIC          MIN
*
* ..
* .. Executable Statements ..
*
* Quick return if possible
*
IF( N.LE.1 )
$  RETURN
*
DO 10 I = 1, NB
  IF( I.GT.1 ) THEN
*
*    Update A(K+1:N,I)
*
*    Update I-th column of A - Y * V**H
*
CALL ZLACGV( I-1, A( K+I-1, 1 ), LDA )
CALL ZGEMV( 'NO TRANSPOSE', N-K, I-1, -ONE, Y(K+1,1), LDY,
$          A( K+I-1, 1 ), LDA, ONE, A( K+1, I ), 1 )
CALL ZLACGV( I-1, A( K+I-1, 1 ), LDA )
*
*    Apply I - V * T**H * V**H to this column (call it b) from the
*    left, using the last column of T as workspace
*
*    Let  V = ( V1 )   and   b = ( b1 )   (first I-1 rows)
*           ( V2 )           ( b2 )
*
*    where V1 is unit lower triangular
*
*    w := V1**H * b1
*
CALL ZCOPY( I-1, A( K+1, I ), 1, T( 1, NB ), 1 )
CALL ZTRMV( 'Lower', 'Conjugate transpose', 'UNIT',
$          I-1, A( K+1, 1 ),
$          LDA, T( 1, NB ), 1 )
*
*    w := w + V2**H * b2
*
CALL ZGEMV( 'Conjugate transpose', N-K-I+1, I-1,
$          ONE, A( K+I, 1 ),
$          LDA, A( K+I, I ), 1, ONE, T( 1, NB ), 1 )
*
*    w := T**H * w
*
CALL ZTRMV( 'Upper', 'Conjugate transpose', 'NON-UNIT',
$          I-1, T, LDT,

```

```

$          T( 1, NB ), 1 )
*
*          b2 := b2 - V2*w
*
*          CALL ZGEMV( 'NO TRANSPOSE', N-K-I+1, I-1, -ONE,
$              A( K+I, 1 ),
$              LDA, T( 1, NB ), 1, ONE, A( K+I, I ), 1 )
*
*          b1 := b1 - V1*w
*
*          CALL ZTRMV( 'Lower', 'NO TRANSPOSE',
$              'UNIT', I-1,
$              A( K+1, 1 ), LDA, T( 1, NB ), 1 )
*          CALL ZAXPY( I-1, -ONE, T( 1, NB ), 1, A( K+1, I ), 1 )
*
*          A( K+I-1, I-1 ) = EI
*          END IF
*
*          Generate the elementary reflector H(I) to annihilate
*          A(K+I+1:N,I)
*
*          CALL ZLARFG( N-K-I+1, A( K+I, I ), A( MIN( K+I+1, N ), I ), 1,
$              TAU( I ) )
*          EI = A( K+I, I )
*          A( K+I, I ) = ONE
*
*          Compute Y(K+1:N,I)
*
*          CALL ZGEMV( 'NO TRANSPOSE', N-K, N-K-I+1,
$              ONE, A( K+1, I+1 ),
$              LDA, A( K+I, I ), 1, ZERO, Y( K+1, I ), 1 )
*          CALL ZGEMV( 'Conjugate transpose', N-K-I+1, I-1,
$              ONE, A( K+I, 1 ), LDA,
$              A( K+I, I ), 1, ZERO, T( 1, I ), 1 )
*          CALL ZGEMV( 'NO TRANSPOSE', N-K, I-1, -ONE,
$              Y( K+1, 1 ), LDY,
$              T( 1, I ), 1, ONE, Y( K+1, I ), 1 )
*          CALL ZSCAL( N-K, TAU( I ), Y( K+1, I ), 1 )
*
*          Compute T(1:I,I)
*
*          CALL ZSCAL( I-1, -TAU( I ), T( 1, I ), 1 )
*          CALL ZTRMV( 'Upper', 'No Transpose', 'NON-UNIT',
$              I-1, T, LDT,
$              T( 1, I ), 1 )
*          T( I, I ) = TAU( I )
*
10 CONTINUE
*          A( K+NB, NB ) = EI
*

```

```

*      Compute Y(1:K,1:NB)
*
      CALL ZLACPY( 'ALL', K, NB, A( 1, 2 ), LDA, Y, LDY )
      CALL ZTRMM( 'RIGHT', 'Lower', 'NO TRANSPOSE',
$           'UNIT', K, NB,
$           ONE, A( K+1, 1 ), LDA, Y, LDY )
      IF( N.GT.K+NB )
$      CALL ZGEMM( 'NO TRANSPOSE', 'NO TRANSPOSE', K,
$           NB, N-K-NB, ONE,
$           A( 1, 2+NB ), LDA, A( K+1+NB, 1 ), LDA, ONE, Y,
$           LDY )
      CALL ZTRMM( 'RIGHT', 'Upper', 'NO TRANSPOSE',
$           'NON-UNIT', K, NB,
$           ONE, T, LDT, Y, LDY )
*
      RETURN
*
*      End of ZLAHR2
*
      END

```

— LAPACK zlahr2 —

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
   (ignorable zero one))
  (defun zlahr2 (n k nb a lda tau t$ ldt y ldy)
    (declare (type (f2cl-lib:integer4) ldy ldt lda nb k n)
     (type (array f2cl-lib:complex16 (*)) y t$ tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
       (y f2cl-lib:complex16 y-%data% y-%offset%))
      (prog ((ei #C(0.0d0 0.0d0)) (i 0))
        (declare (type (f2cl-lib:complex16) ei) (type (f2cl-lib:integer4) i))
        (if (<= n 1) (go end_label))
        (f2cl-lib:fd0 (i 1 (f2cl-lib:int-add i 1))
          ((> i nb) nil)
          (tagbody
            (cond
              ((> i 1)
               (zlacgv (f2cl-lib:int-sub i 1)
                (f2cl-lib:array-slice a-%data% f2cl-lib:complex16

```



```

      ((+ k i (f2cl-lib:int-sub 1)) 1) ((1 lda) (1 *))
      a-%offset%)
lda)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (zgemv "NO TRANSPOSE"
  (f2cl-lib:int-sub n k) (f2cl-lib:int-sub i 1)
  (- one)
  (f2cl-lib:array-slice y-%data% f2cl-lib:complex16
   ((+ k 1) 1)
   ((1 ldy) (1 nb)) y-%offset%)
  ldy
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
   ((+ k i (f2cl-lib:int-sub 1)) 1) ((1 lda) (1 *))
   a-%offset%)
  lda one
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
   ((+ k 1) i)
   ((1 lda) (1 *)) a-%offset%)
  1)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6
  var-9 var-10))
 (when var-5 (setf ldy var-5))
 (when var-7 (setf lda var-7))
 (when var-8 (setf one var-8)))
(zlacgv (f2cl-lib:int-sub i 1)
 (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
  ((+ k i (f2cl-lib:int-sub 1)) 1) ((1 lda) (1 *))
  a-%offset%)
lda)
(zcopy (f2cl-lib:int-sub i 1)
 (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
  ((+ k 1) i)
  ((1 lda) (1 *)) a-%offset%)
1)
(f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
 (1 nb) ((1 ldt) (1 nb))
 t$-%offset%)
1)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4
  var-5 var-6 var-7)
 (ztrmv "Lower" "Conjugate transpose" "UNIT"
  (f2cl-lib:int-sub i 1)
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
   ((+ k 1) 1)
   ((1 lda) (1 *)) a-%offset%)
  lda
  (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 nb)
   ((1 ldt) (1 nb)) t$-%offset%)

```

```

1)
(declare (ignore var-0 var-1 var-2 var-3 var-4
            var-6 var-7))
(when var-5 (setf lda var-5)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (zgemv "Conjugate transpose"
    (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
    (f2cl-lib:int-sub i 1) one
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ k i) 1)
      ((1 lda) (1 *)) a-%offset%)
    lda
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ k i) i)
      ((1 lda) (1 *)) a-%offset%)
    1 one
    (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 nb)
      ((1 ldt) (1 nb)) t$-%offset%)
  1)
(declare (ignore var-0 var-1 var-2 var-4 var-6
            var-7 var-9 var-10))
(when var-3 (setf one var-3))
(when var-5 (setf lda var-5))
(when var-8 (setf one var-8))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4
  var-5 var-6 var-7)
  (ztrmv "Upper" "Conjugate transpose" "NON-UNIT"
    (f2cl-lib:int-sub i 1) t$
    ldt
    (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 nb)
      ((1 ldt) (1 nb)) t$-%offset%)
  1)
(declare (ignore var-0 var-1 var-2 var-3 var-4
            var-6 var-7))
(when var-5 (setf ldt var-5)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (zgemv "NO TRANSPOSE"
    (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
    (f2cl-lib:int-sub i 1) (- one)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ k i) 1)
      ((1 lda) (1 *)) a-%offset%)
    lda
    (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 nb)
      ((1 ldt) (1 nb)) t$-%offset%)
  1 one

```

```

(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
  ((+ k i) i)
  ((1 lda) (1 *)) a-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-6
  var-7 var-9 var-10))
(when var-5 (setf lda var-5))
(when var-8 (setf one var-8))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4
  var-5 var-6 var-7)
  (ztrmv "Lower" "NO TRANSPOSE" "UNIT"
    (f2cl-lib:int-sub i 1)
    (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
      ((+ k 1) 1)
      ((1 lda) (1 *)) a-%offset%)
    lda
    (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 nb)
      ((1 ldt) (1 nb)) t$-%offset%)
    1)
  (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6
    var-7))
  (when var-5 (setf lda var-5)))
(zaxpy (f2cl-lib:int-sub i 1) (- one)
  (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
    (1 nb) ((1 ldt) (1 nb))
    t$-%offset%)
  1
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    ((+ k 1) i)
    ((1 lda) (1 *)) a-%offset%)
  1)
(setf
  (f2cl-lib:fref a-%data%
    ((f2cl-lib:int-sub (f2cl-lib:int-add k i) 1)
    (f2cl-lib:int-sub i 1))
    ((1 lda) (1 *)) a-%offset%)
  ei)))
(zlarfg (f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    ((+ k i) i)
    ((1 lda) (1 *)) a-%offset%)
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    ((min (f2cl-lib:int-add k i 1) n) i) ((1 lda) (1 *))
    a-%offset%)
  1
  (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
    (i) ((1 nb))
    tau-%offset%))
(setf ei
  (f2cl-lib:fref a-%data% ((f2cl-lib:int-add k i) i)

```

```

((1 lda) (1 *))
a-%offset%)
(setf
(f2cl-lib:fref a-%data% ((f2cl-lib:int-add k i) i)
((1 lda) (1 *))
a-%offset%)
one)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
var-9 var-10)
(zgemv "NO TRANSPOSE" (f2cl-lib:int-sub n k)
(f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1) one
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
((+ k 1) (f2cl-lib:int-add i 1)) ((1 lda) (1 *)) a-%offset%)
lda
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
((+ k i) i)
((1 lda) (1 *)) a-%offset%)
1 zero
(f2cl-lib:array-slice y-%data% f2cl-lib:complex16
((+ k 1) i)
((1 ldy) (1 nb)) y-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-4 var-6 var-7
var-9 var-10))
(when var-3 (setf one var-3)) (when var-5 (setf lda var-5))
(when var-8 (setf zero var-8)))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
var-9 var-10)
(zgemv "Conjugate transpose"
(f2cl-lib:int-add (f2cl-lib:int-sub n k i) 1)
(f2cl-lib:int-sub i 1) one
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
((+ k i) 1)
((1 lda) (1 *)) a-%offset%)
lda
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
((+ k i) i)
((1 lda) (1 *)) a-%offset%)
1 zero
(f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
(1 i) ((1 ldt) (1 nb))
t$-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-4 var-6 var-7
var-9 var-10))
(when var-3 (setf one var-3)) (when var-5 (setf lda var-5))
(when var-8 (setf zero var-8)))
(multiple-value-bind

```

```

(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
 var-9 var-10)
(zgemv "NO TRANSPOSE" (f2cl-lib:int-sub n k)
      (f2cl-lib:int-sub i 1) (- one)
 (f2cl-lib:array-slice y-%data% f2cl-lib:complex16
  ((+ k 1) 1)
  ((1 ldy) (1 nb)) y-%offset%)
ldy
(f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
  (1 i) ((1 ldt) (1 nb))
  t$-%offset%)
1 one
(f2cl-lib:array-slice y-%data% f2cl-lib:complex16
  ((+ k 1) i)
  ((1 ldy) (1 nb)) y-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-3 var-4 var-6
 var-7 var-9 var-10))
(when var-5 (setf ldy var-5)) (when var-8 (setf one var-8)))
(zscal (f2cl-lib:int-sub n k)
 (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
  (i) ((1 nb))
  tau-%offset%)
 (f2cl-lib:array-slice y-%data% f2cl-lib:complex16
  ((+ k 1) i)
  ((1 ldy) (1 nb)) y-%offset%)
1)
(zscal (f2cl-lib:int-sub i 1)
 (- (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
 (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
  (1 i) ((1 ldt) (1 nb))
  t$-%offset%)
1)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
 (ztrmv "Upper" "No Transpose" "NON-UNIT"
  (f2cl-lib:int-sub i 1) t$ ldt
  (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
   (1 i) ((1 ldt) (1 nb))
   t$-%offset%)
 1)
 (declare (ignore var-0 var-1 var-2 var-3 var-4 var-6 var-7))
 (when var-5 (setf ldt var-5)))
(setf (f2cl-lib:fref t$-%data% (i i) ((1 ldt) (1 nb))
  t$-%offset%)
 (f2cl-lib:fref tau-%data% (i) ((1 nb)) tau-%offset%))
label10))
(setf
 (f2cl-lib:fref a-%data% ((f2cl-lib:int-add k nb) nb) ((1 lda) (1 *)))
 a-%offset%)

```

```

ei)
(zlacpy "ALL" k nb
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
  (1 2) ((1 lda) (1 *))
  a-%offset%)
lda y ldy)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9 var-10)
(ztrmm "RIGHT" "Lower" "NO TRANSPOSE" "UNIT" k nb one
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16 ((+ k 1) 1)
  ((1 lda) (1 *)) a-%offset%)
lda y ldy)
(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf k var-4)) (when var-5 (setf nb var-5))
(when var-6 (setf one var-6)) (when var-8 (setf lda var-8))
(when var-10 (setf ldy var-10)))
(if (> n (f2cl-lib:int-add k nb))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
var-10 var-11
var-12)
(zgemm "NO TRANSPOSE" "NO TRANSPOSE"
  k nb (f2cl-lib:int-sub n k nb) one
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16
  (1 (f2cl-lib:int-add 2 nb)) ((1 lda) (1 *)) a-%offset%)
lda
(f2cl-lib:array-slice a-%data% f2cl-lib:complex16 ((+ k 1 nb) 1)
  ((1 lda) (1 *)) a-%offset%)
lda one y ldy)
(declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
(when var-2 (setf k var-2)) (when var-3 (setf nb var-3))
(when var-5 (setf one var-5)) (when var-7 (setf lda var-7))
(when var-9 (setf lda var-9)) (when var-10 (setf one var-10))
(when var-12 (setf ldy var-12))))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9 var-10)
(ztrmm "RIGHT" "Upper" "NO TRANSPOSE" "NON-UNIT" k nb one
  t$ ldt y ldy)
(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf k var-4)) (when var-5 (setf nb var-5))
(when var-6 (setf one var-6)) (when var-8 (setf ldt var-8))
(when var-10 (setf ldy var-10)))
(go end_label) end_label
(return (values nil k nb nil lda nil nil ldt nil ldy))))))

```

zlange LAPACK**— zlange.input —**

```

)set break resume
)sys rm -f zlange.output
)spool zlange.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlange.help —

```

=====
zlange examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

DOUBLE PRECISION FUNCTION ZLANGE(NORM, M, N, A, LDA, WORK)

```

.. Scalar Arguments ..
CHARACTER          NORM
INTEGER            LDA, M, N
..
.. Array Arguments ..
DOUBLE PRECISION   WORK( * )
COMPLEX*16         A( LDA, * )
..

```

Purpose:
 =====

ZLANGE returns the value of the one norm, or the Frobenius norm, or

the infinity norm, or the element of largest absolute value of a complex matrix A.

```
ZLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
          (
            ( norm1(A),          NORM = '1', 'O' or 'o'
              (
                ( normI(A),       NORM = 'I' or 'i'
                  (
                    ( normF(A),    NORM = 'F', 'f', 'E' or 'e'
```

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Arguments:
=====

[in] NORM

NORM is CHARACTER*1
Specifies the value to be returned in ZLANGE as described above.

[in] M

M is INTEGER
The number of rows of the matrix A. $M \geq 0$. When $M = 0$, ZLANGE is set to zero.

[in] N

N is INTEGER
The number of columns of the matrix A. $N \geq 0$. When $N = 0$, ZLANGE is set to zero.

[in] A

A is COMPLEX*16 array, dimension (LDA,N)
The m by n matrix A.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. $LDA \geq \max(M,1)$.

[out] WORK

WORK is DOUBLE PRECISION array, dimension (MAX(1,LWORK)),
 where LWORK >= M when NORM = 'I'; otherwise, WORK is not
 referenced.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlange.f —

```
* =====
*      DOUBLE PRECISION FUNCTION ZLANGE( NORM, M, N, A, LDA, WORK )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER           NORM
*      INTEGER             LDA, M, N
*
*      ..
*      .. Array Arguments ..
*      DOUBLE PRECISION    WORK( * )
*      COMPLEX*16          A( LDA, * )
*
*      ..
*
* =====
*
*      .. Parameters ..
*      DOUBLE PRECISION    ONE, ZERO
*      PARAMETER           ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      ..
*      .. Local Scalars ..
*      INTEGER             I, J
*      DOUBLE PRECISION    SCALE, SUM, VALUE
*
*      ..
*      .. External Functions ..
*      LOGICAL             LSAME
*      EXTERNAL            LSAME
```

```

*      ..
*      .. External Subroutines ..
      EXTERNAL          ZLASSQ
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, MIN, SQRT
*      ..
*      .. Executable Statements ..
*
      IF( MIN( M, N ).EQ.0 ) THEN
        VALUE = ZERO
      ELSE IF( LSAME( NORM, 'M' ) ) THEN
*
*        Find max(abs(A(i,j))).
*
        VALUE = ZERO
        DO 20 J = 1, N
          DO 10 I = 1, M
            VALUE = MAX( VALUE, ABS( A( I, J ) ) )
10          CONTINUE
20        CONTINUE
      ELSE IF( ( LSAME( NORM, 'O' ) ) .OR. ( NORM.EQ.'1' ) ) THEN
*
*        Find norm1(A).
*
        VALUE = ZERO
        DO 40 J = 1, N
          SUM = ZERO
          DO 30 I = 1, M
            SUM = SUM + ABS( A( I, J ) )
30          CONTINUE
          VALUE = MAX( VALUE, SUM )
40        CONTINUE
      ELSE IF( LSAME( NORM, 'I' ) ) THEN
*
*        Find normI(A).
*
        DO 50 I = 1, M
          WORK( I ) = ZERO
50        CONTINUE
        DO 70 J = 1, N
          DO 60 I = 1, M
            WORK( I ) = WORK( I ) + ABS( A( I, J ) )
60          CONTINUE
70        CONTINUE
        VALUE = ZERO
        DO 80 I = 1, M
          VALUE = MAX( VALUE, WORK( I ) )
80        CONTINUE
      ELSE IF( ( LSAME( NORM, 'F' ) ) .OR. ( LSAME( NORM, 'E' ) ) ) THEN

```

```

*
*      Find normF(A).
*
      SCALE = ZERO
      SUM = ONE
      DO 90 J = 1, N
          CALL ZLASSQ( M, A( 1, J ), 1, SCALE, SUM )
90    CONTINUE
      VALUE = SCALE*SQRT( SUM )
      END IF
*
      ZLANGE = VALUE
      RETURN
*
*      End of ZLANGE
*
      END

```

— LAPACK zlange —

```

(let* ((one 1.0) (zero 0.0))
  (declare (type (double-float 1.0 1.0) one)
            (type (double-float 0.0 0.0) zero))
  (defun zlange (norm m n a lda work)
    (declare (type (simple-array double-float (*)) work)
              (type (simple-array (complex double-float) (*)) a)
              (type fixnum lda n m)
              (type character norm))
    (f2cl-lib:with-multi-array-data
      ((norm character norm-%data% norm-%offset%)
       (a (complex double-float) a-%data% a-%offset%)
       (work double-float work-%data% work-%offset%))
      (prog ((scale 0.0) (sum 0.0) (value 0.0) (i 0) (j 0) (zlange 0.0))
        (declare (type fixnum i j)
                  (type (double-float) scale sum value zlange))
        (cond
          ((= (min (the fixnum m) (the fixnum n)) 0)
            (setf value zero))
          ((char-equal norm #\M)
            (setf value zero)
            (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                          (> j n) nil)
            (tagbody
              (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                            (> i m) nil)
              (tagbody

```

```

      (setf value
        (max value
          (abs
            (f2cl-lib:fref a-%data%
                          (i j)
                          ((1 lda) (1 *))
                          a-%offset%))))))
    ((or (char-equal norm #\0) (f2cl-lib:fstring= norm "1"))
     (setf value zero)
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (setf sum zero)
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
      (tagbody
       (setf sum
         (+ sum
           (abs
             (f2cl-lib:fref a-%data%
                           (i j)
                           ((1 lda) (1 *))
                           a-%offset%))))))
       (setf value (max value sum))))
    ((char-equal norm #\I)
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                   ((> i m) nil)
     (tagbody
      (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
            zero))
     (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                   ((> j n) nil)
     (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                    ((> i m) nil)
      (tagbody
       (setf (f2cl-lib:fref work-%data% (i) ((1 *)) work-%offset%)
             (+
              (f2cl-lib:fref work-%data%
                              (i)
                              ((1 *))
                              work-%offset%)
              (abs
                (f2cl-lib:fref a-%data%
                              (i j)
                              ((1 lda) (1 *))
                              a-%offset%))))))
       (setf value zero)
       (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                     ((> i m) nil)

```

```

      (tagbody
        (setf value
          (max value
            (f2cl-lib:fref work-%data%
                          (i)
                          ((1 *))
                          work-%offset%))))))
      ((or (char-equal norm #\F) (char-equal norm #\E))
        (setf scale zero)
        (setf sum one)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j n) nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
            (zlassq m
              (f2cl-lib:array-slice a
                                    (complex double-float)
                                    (1 j)
                                    ((1 lda) (1 *)))
              1 scale sum)
            (declare (ignore var-0 var-1 var-2))
            (setf scale var-3)
            (setf sum var-4))))
        (setf value (* scale (f2cl-lib:fsqrt sum))))
      (setf zlange value)
      (return (values zlange nil nil nil nil nil nil))))))

```

zlaqr0 LAPACK

— zlaqr0.input —

```

)set break resume
)sys rm -f zlaqr0.output
)spool zlaqr0.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlaqr0.help —

```
=====
zlaqr0 examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====

SUBROUTINE ZLAQRO( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
                  IHIZ, Z, LDZ, WORK, LWORK, INFO )

.. Scalar Arguments ..
INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, LWORK, N
LOGICAL          WANTT, WANTZ
..
.. Array Arguments ..
COMPLEX*16       H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
..
```

Purpose:

```
=====
```

ZLAQRO computes the eigenvalues of a Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z T Z^* H$, where T is an upper triangular matrix (the Schur form), and Z is the unitary matrix of Schur vectors.

Optionally Z may be postmultiplied into an input unitary matrix Q so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the unitary matrix Q: $A = Q H Q^* H = (QZ) H (QZ)^* H$.

Arguments:

```
=====
```

[in] WANTT

WANTT is LOGICAL
 = .TRUE. : the full Schur form T is required;
 = .FALSE.: only eigenvalues are required.

[in] WANTZ

WANTZ is LOGICAL
 = .TRUE. : the matrix of Schur vectors Z is required;
 = .FALSE.: Schur vectors are not required.

[in] N

N is INTEGER
 The order of the matrix H. N .GE. 0.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by ZGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)
 On entry, the upper Hessenberg matrix H.
 On exit, if INFO = 0 and WANTT is .TRUE., then H contains the upper triangular matrix T from the Schur decomposition (the Schur form). If INFO = 0 and WANT is .FALSE., then the contents of H are unspecified on exit. (The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set $H(i,j) = 0$ for $i.GT.j$ and $j = 1, 2, \dots, ILO-1$ or $j = IHI+1, IHI+2, \dots, N$.

[in] LDH

LDH is INTEGER
 The leading dimension of the array H. LDH .GE. max(1,N).

[out] W

W is COMPLEX*16 array, dimension (N)
 The computed eigenvalues of $H(ILO:IHI, ILO:IHI)$ are stored in $W(ILO:IHI)$. If WANTT is .TRUE., then the eigenvalues are

stored in the same order as on the diagonal of the Schur form returned in H, with $W(i) = H(i,i)$.

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE..

1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,IHI)

If WANTZ is .FALSE., then Z is not referenced.

If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by $Z(ILO:IHI,ILOZ:IHIZ)*U$ where U is the orthogonal Schur factor of $H(ILO:IHI,ILO:IHI)$.

(The output value of Z when INFO.GT.0 is given under the description of INFO below.)

[in] LDZ

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then $LDZ \geq \max(1, IHIZ)$. Otherwise, $LDZ \geq 1$.

[out] WORK

WORK is COMPLEX*16 array, dimension LWORK

On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

[in] LWORK

LWORK is INTEGER

The dimension of the array WORK. LWORK $\geq \max(1, N)$ is sufficient, but LWORK typically as large as $6*N$ may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then ZLAQR0 does a workspace query. In this case, ZLAQR0 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

[out] INFO

INFO is INTEGER

= 0: successful exit

.GT. 0: if INFO = i, ZLAQRO failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit, the remaining unconverged eigenvalues are the eigenvalues of the upper Hessenberg matrix rows and columns ILO through INFO of the final, output value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final value of H is upper Hessenberg and triangular in rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ)
= (initial value of Z(ILO:IHI,ILOZ:IHIZ))*U

where U is the unitary matrix in (*) (regardless of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not accessed.

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Contributors:

=====

Karen Braman and Ralph Byers, Department of Mathematics,
University of Kansas, USA

References:

=====

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR Algorithm Part I: Maintaining Well Focused Shifts, and Level 3 Performance, SIAM Journal of Matrix Analysis, volume 23, pages 929--947, 2002.

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR Algorithm Part II: Aggressive Early Deflation, SIAM Journal of Matrix Analysis, volume 23, pages 948--973, 2002.

— zlaqr0.f —

```
* =====
*      SUBROUTINE ZLAQRO( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
*      $                IHIZ, Z, LDZ, WORK, LWORK, INFO )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, LWORK, N
*      LOGICAL          WANTT, WANTZ
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
*      ..
*
*      =====
*
*      .. Parameters ..
*
*      ==== Matrices of order NTINY or smaller must be processed by
*      .    ZLAHQR because of insufficient subdiagonal scratch space.
*      .    (This is a hard limit.) ====
*      INTEGER          NTINY
*      PARAMETER        ( NTINY = 11 )
*
*      ==== Exceptional deflation windows: try to cure rare
*      .    slow convergence by varying the size of the
*      .    deflation window after KEXNW iterations. ====
*      INTEGER          KEXNW
```

```

      PARAMETER          ( KEXNW = 5 )
*
*
*   ==== Exceptional shifts: try to cure rare slow convergence
*   .   with ad-hoc exceptional shifts every KEXSH iterations.
*   .   ====
      INTEGER            KEXSH
      PARAMETER          ( KEXSH = 6 )
*
*
*   ==== The constant WILK1 is used to form the exceptional
*   .   shifts. ====
      DOUBLE PRECISION   WILK1
      PARAMETER          ( WILK1 = 0.75d0 )
      COMPLEX*16          ZERO, ONE
      PARAMETER          ( ZERO = ( 0.0d0, 0.0d0 ),
$                          ONE = ( 1.0d0, 0.0d0 ) )
      DOUBLE PRECISION   TWO
      PARAMETER          ( TWO = 2.0d0 )
*
*   ..
*   .. Local Scalars ..
      COMPLEX*16          AA, BB, CC, CDUM, DD, DET, RTDISC, SWAP, TR2
      DOUBLE PRECISION    S
      INTEGER             I, INF, IT, ITMAX, K, KACC22, KBOT, KDU, KS,
$                        KT, KTOP, KU, KV, KWH, KWTOP, KWV, LD, LS,
$                        LWKOPT, NDEC, NDFL, NH, NHO, NIBBLE, NMIN, NS,
$                        NSMAX, NSR, NVE, NW, NWMAX, NWR, NWUPBD
      LOGICAL             SORTED
      CHARACTER           JBCMPZ*2
*
*   ..
*   .. External Functions ..
      INTEGER             ILAENV
      EXTERNAL             ILAENV
*
*   ..
*   .. Local Arrays ..
      COMPLEX*16          ZDUM( 1, 1 )
*
*   ..
*   .. External Subroutines ..
      EXTERNAL             ZLACPY, ZLAHQR, ZLAQR3, ZLAQR4, ZLAQR5
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC            ABS, DBLE, DCMPLX, DIMAG, INT, MAX, MIN, MOD,
$                          SQR
*
*   ..
*   .. Statement Functions ..
      DOUBLE PRECISION    CABS1
*
*   ..
*   .. Statement Function definitions ..
      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
*   ..
*   .. Executable Statements ..
      INFO = 0

```

```

*
*      ==== Quick return for N = 0: nothing to do. ====
*
      IF( N.EQ.0 ) THEN
        WORK( 1 ) = ONE
        RETURN
      END IF
*
      IF( N.LE.NTINY ) THEN
*
*      ==== Tiny matrices must use ZLAHQR. ====
*
        LWKOPT = 1
        IF( LWORK.NE.-1 )
$          CALL ZLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
$                      IHIZ, Z, LDZ, INFO )
        ELSE
*
*      ==== Use small bulge multi-shift QR with aggressive early
*      .      deflation on larger-than-tiny matrices. ====
*
*      ==== Hope for the best. ====
*
        INFO = 0
*
*      ==== Set up job flags for ILAENV. ====
*
        IF( WANTT ) THEN
          JBCMPZ( 1: 1 ) = 'S'
        ELSE
          JBCMPZ( 1: 1 ) = 'E'
        END IF
        IF( WANTZ ) THEN
          JBCMPZ( 2: 2 ) = 'V'
        ELSE
          JBCMPZ( 2: 2 ) = 'N'
        END IF
*
*      ==== NWR = recommended deflation window size. At this
*      .      point, N .GT. NTINY = 11, so there is enough
*      .      subdiagonal workspace for NWR.GE.2 as required.
*      .      (In fact, there is enough subdiagonal space for
*      .      NWR.GE.3.) ====
*
        NWR = ILAENV( 13, 'ZLAQRO', JBCMPZ, N, ILO, IHI, LWORK )
        NWR = MAX( 2, NWR )
        NWR = MIN( IHI-ILO+1, ( N-1 ) / 3, NWR )
*
*      ==== NSR = recommended number of simultaneous shifts.
*      .      At this point N .GT. NTINY = 11, so there is at

```

```

*      .      enough subdiagonal workspace for NSR to be even
*      .      and greater than or equal to two as required. ====
*
      NSR = ILAENV( 15, 'ZLAQR0', JBCMPZ, N, ILO, IHI, LWORK )
      NSR = MIN( NSR, ( N+6 ) / 9, IHI-ILO )
      NSR = MAX( 2, NSR-MOD( NSR, 2 ) )
*
*      ==== Estimate optimal workspace ====
*
*      ==== Workspace query call to ZLAQR3 ====
*
      CALL ZLAQR3( WANTT, WANTZ, N, ILO, IHI, NWR+1, H, LDH, ILOZ,
$              IHIZ, Z, LDZ, LS, LD, W, H, LDH, N, H, LDH, N, H,
$              LDH, WORK, -1 )
*
*      ==== Optimal workspace = MAX(ZLAQR5, ZLAQR3) ====
*
      LWKOPT = MAX( 3*NSR / 2, INT( WORK( 1 ) ) )
*
*      ==== Quick return in case of workspace query. ====
*
      IF( LWORK.EQ.-1 ) THEN
          WORK( 1 ) = DCMPLX( LWKOPT, 0 )
          RETURN
      END IF
*
*      ==== ZLAHQR/ZLAQR0 crossover point ====
*
      NMN = ILAENV( 12, 'ZLAQR0', JBCMPZ, N, ILO, IHI, LWORK )
      NMN = MAX( NTINY, NMN )
*
*      ==== Nibble crossover point ====
*
      NIBBLE = ILAENV( 14, 'ZLAQR0', JBCMPZ, N, ILO, IHI, LWORK )
      NIBBLE = MAX( 0, NIBBLE )
*
*      ==== Accumulate reflections during ttswp? Use block
*      .      2-by-2 structure during matrix-matrix multiply? ====
*
      KACC22 = ILAENV( 16, 'ZLAQR0', JBCMPZ, N, ILO, IHI, LWORK )
      KACC22 = MAX( 0, KACC22 )
      KACC22 = MIN( 2, KACC22 )
*
*      ==== NWMAX = the largest possible deflation window for
*      .      which there is sufficient workspace. ====
*
      NWMAX = MIN( ( N-1 ) / 3, LWORK / 2 )
      NW = NWMAX
*
*      ==== NSMAX = the Largest number of simultaneous shifts

```

```

*      .   for which there is sufficient workspace. ====
*
      NSMAX = MIN( ( N+6 ) / 9, 2*LWORK / 3 )
      NSMAX = NSMAX - MOD( NSMAX, 2 )
*
*      ==== NDFL: an iteration count restarted at deflation. ====
*
      NDFL = 1
*
*      ==== ITMAX = iteration limit ====
*
      ITMAX = MAX( 30, 2*KEXSH )*MAX( 10, ( IHI-ILO+1 ) )
*
*      ==== Last row and column in the active block ====
*
      KBOT = IHI
*
*      ==== Main Loop ====
*
      DO 70 IT = 1, ITMAX
*
*          ==== Done when KBOT falls below ILO ====
*
*          IF( KBOT.LT.ILO )
$              GO TO 80
*
*          ==== Locate active block ====
*
*          DO 10 K = KBOT, ILO + 1, -1
*              IF( H( K, K-1 ).EQ.ZERO )
$                  GO TO 20
10          CONTINUE
*              K = ILO
20          CONTINUE
*              KTOP = K
*
*          ==== Select deflation window size:
*
*          .   Typical Case:
*          .   If possible and advisable, nibble the entire
*          .   active block. If not, use size MIN(NWR,NWMAX)
*          .   or MIN(NWR+1,NWMAX) depending upon which has
*          .   the smaller corresponding subdiagonal entry
*          .   (a heuristic).
*          .
*          .   Exceptional Case:
*          .   If there have been no deflations in KEXNW or
*          .   more iterations, then vary the deflation window
*          .   size. At first, because, larger windows are,
*          .   in general, more powerful than smaller ones,
*          .   rapidly increase the window to the maximum possible.

```

```

*          .      Then, gradually reduce the window size. ====
*
      NH = KBOT - KTOP + 1
      NWUPBD = MIN( NH, NWMAX )
      IF( NDFL.LT.KEXNW ) THEN
        NW = MIN( NWUPBD, NWR )
      ELSE
        NW = MIN( NWUPBD, 2*NW )
      END IF
      IF( NW.LT.NWMAX ) THEN
        IF( NW.GE.NH-1 ) THEN
          NW = NH
        ELSE
          KWTOP = KBOT - NW + 1
          IF( CABS1( H( KWTOP, KWTOP-1 ) ).GT.
$          CABS1( H( KWTOP-1, KWTOP-2 ) ) )NW = NW + 1
          END IF
        END IF
      IF( NDFL.LT.KEXNW ) THEN
        NDEC = -1
      ELSE IF( NDEC.GE.0 .OR. NW.GE.NWUPBD ) THEN
        NDEC = NDEC + 1
        IF( NW-NDEC.LT.2 )
$          NDEC = 0
        NW = NW - NDEC
      END IF

*
*      ==== Aggressive early deflation:
*      .      split workspace under the subdiagonal into
*      .      - an nw-by-nw work array V in the lower
*      .      left-hand-corner,
*      .      - an NW-by-at-least-NW-but-more-is-better
*      .      (NW-by-NHO) horizontal work array along
*      .      the bottom edge,
*      .      - an at-least-NW-but-more-is-better (NHV-by-NW)
*      .      vertical work array along the left-hand-edge.
*      .      ====
*
      KV = N - NW + 1
      KT = NW + 1
      NHO = ( N-NW-1 ) - KT + 1
      KWV = NW + 2
      NVE = ( N-NW ) - KWV + 1

*
*      ==== Aggressive early deflation ====
*
      CALL ZLAQR3( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
$              IHIZ, Z, LDZ, LS, LD, W, H( KV, 1 ), LDH, NHO,
$              H( KV, KT ), LDH, NVE, H( KWV, 1 ), LDH, WORK,
$              LWORK )

```

```

*
*      ==== Adjust KBOT accounting for new deflations. ====
*
*      KBOT = KBOT - LD
*
*      ==== KS points to the shifts. ====
*
*      KS = KBOT - LS + 1
*
*      ==== Skip an expensive QR sweep if there is a (partly
*      . heuristic) reason to expect that many eigenvalues
*      . will deflate without it. Here, the QR sweep is
*      . skipped if many eigenvalues have just been deflated
*      . or if the remaining active block is small.
*
*      IF( ( LD.EQ.0 ) .OR. ( ( 100*LD.LE.NW*NIBBLE ) .AND. ( KBOT-
$      KTOP+1.GT.MIN( NMIN, NWMAX ) ) ) ) THEN
*
*      ==== NS = nominal number of simultaneous shifts.
*      . This may be lowered (slightly) if ZLAQR3
*      . did not provide that many shifts. ====
*
*      NS = MIN( NSMAX, NSR, MAX( 2, KBOT-KTOP ) )
*      NS = NS - MOD( NS, 2 )
*
*      ==== If there have been no deflations
*      . in a multiple of KEXSH iterations,
*      . then try exceptional shifts.
*      . Otherwise use shifts provided by
*      . ZLAQR3 above or from the eigenvalues
*      . of a trailing principal submatrix. ====
*
*      IF( MOD( NDFL, KEXSH ).EQ.0 ) THEN
*          KS = KBOT - NS + 1
*          DO 30 I = KBOT, KS + 1, -2
*              W( I ) = H( I, I ) + WILK1*CABS1( H( I, I-1 ) )
*              W( I-1 ) = W( I )
30          CONTINUE
*          ELSE
*
*      ==== Got NS/2 or fewer shifts? Use ZLAQR4 or
*      . ZLAHQR on a trailing principal submatrix to
*      . get more. (Since NS.LE.NSMAX.LE.(N+6)/9,
*      . there is enough space below the subdiagonal
*      . to fit an NS-by-NS scratch array.) ====
*
*      IF( KBOT-KS+1.LE.NS / 2 ) THEN
*          KS = KBOT - NS + 1
*          KT = N - NS + 1
*          CALL ZLACPY( 'A', NS, NS, H( KS, KS ), LDH,

```



```

$           H( KT, 1 ), LDH )
IF( NS.GT.NMIN ) THEN
    CALL ZLAQR4( .false., .false., NS, 1, NS,
$           H( KT, 1 ), LDH, W( KS ), 1, 1,
$           ZDUM, 1, WORK, LWORK, INF )
ELSE
    CALL ZLAHQR( .false., .false., NS, 1, NS,
$           H( KT, 1 ), LDH, W( KS ), 1, 1,
$           ZDUM, 1, INF )
END IF
KS = KS + INF

*
*      ==== In case of a rare QR failure use
*      .   eigenvalues of the trailing 2-by-2
*      .   principal submatrix.  Scale to avoid
*      .   overflows, underflows and subnormals.
*      .   (The scale factor S can not be zero,
*      .   because H(KBOT,KBOT-1) is nonzero.) ====
*
IF( KS.GE.KBOT ) THEN
    S = CABS1( H( KBOT-1, KBOT-1 ) ) +
$       CABS1( H( KBOT, KBOT-1 ) ) +
$       CABS1( H( KBOT-1, KBOT ) ) +
$       CABS1( H( KBOT, KBOT ) )
    AA = H( KBOT-1, KBOT-1 ) / S
    CC = H( KBOT, KBOT-1 ) / S
    BB = H( KBOT-1, KBOT ) / S
    DD = H( KBOT, KBOT ) / S
    TR2 = ( AA+DD ) / TWO
    DET = ( AA-TR2 )*( DD-TR2 ) - BB*CC
    RTDISC = SQRT( -DET )
    W( KBOT-1 ) = ( TR2+RTDISC )*S
    W( KBOT ) = ( TR2-RTDISC )*S

*
    KS = KBOT - 1
END IF
END IF

*
IF( KBOT-KS+1.GT.NS ) THEN

*
*      ==== Sort the shifts (Helps a little) ====
*
SORTED = .false.
DO 50 K = KBOT, KS + 1, -1
    IF( SORTED )
$       GO TO 60
    SORTED = .true.
    DO 40 I = KS, K - 1
        IF( CABS1( W( I ) ).LT.CABS1( W( I+1 ) ) )
$           THEN

```

```

        SORTED = .false.
        SWAP = W( I )
        W( I ) = W( I+1 )
        W( I+1 ) = SWAP
    END IF
40      CONTINUE
50      CONTINUE
60      CONTINUE
    END IF
END IF

*
*      ==== If there are only two shifts, then use
*      .      only one.  ====
*
    IF( KBOT-KS+1.EQ.2 ) THEN
        IF( CABS1( W( KBOT )-H( KBOT, KBOT ) ).LT.
$         CABS1( W( KBOT-1 )-H( KBOT, KBOT ) ) ) THEN
            W( KBOT-1 ) = W( KBOT )
        ELSE
            W( KBOT ) = W( KBOT-1 )
        END IF
    END IF
END IF

*
*      ==== Use up to NS of the the smallest magnatiude
*      .      shifts.  If there aren't NS shifts available,
*      .      then use them all, possibly dropping one to
*      .      make the number of shifts even.  ====
*
    NS = MIN( NS, KBOT-KS+1 )
    NS = NS - MOD( NS, 2 )
    KS = KBOT - NS + 1

*
*      ==== Small-bulge multi-shift QR sweep:
*      .      split workspace under the subdiagonal into
*      .      - a KDU-by-KDU work array U in the lower
*      .      left-hand-corner,
*      .      - a KDU-by-at-least-KDU-but-more-is-better
*      .      (KDU-by-NHo) horizontal work array WH along
*      .      the bottom edge,
*      .      - and an at-least-KDU-but-more-is-better-by-KDU
*      .      (NVE-by-KDU) vertical work WV array along
*      .      the left-hand-edge.  ====
*
    KDU = 3*NS - 3
    KU = N - KDU + 1
    KWH = KDU + 1
    NHO = ( N-KDU+1-4 ) - ( KDU+1 ) + 1
    KWV = KDU + 4
    NVE = N - KDU - KWV + 1
*

```

```

*          ==== Small-bulge multi-shift QR sweep ====
*
*          CALL ZLAQR5( WANTT, WANTZ, KACC22, N, KTOP, KBOT, NS,
$              W( KS ), H, LDH, ILOZ, IHIZ, Z, LDZ, WORK,
$              3, H( KU, 1 ), LDH, NVE, H( KWV, 1 ), LDH,
$              NHO, H( KU, KWH ), LDH )
*          END IF
*
*          ==== Note progress (or the lack of it). ====
*
*          IF( LD.GT.0 ) THEN
*              NDFL = 1
*          ELSE
*              NDFL = NDFL + 1
*          END IF
*
*          ==== End of main loop ====
70  CONTINUE
*
*          ==== Iteration limit exceeded. Set INFO to show where
*          .   the problem occurred and exit. ====
*
*          INFO = KBOT
80  CONTINUE
*          END IF
*
*          ==== Return the optimal value of LWORK. ====
*
*          WORK( 1 ) = DCMPLX( LWKOPT, 0 )
*
*          ==== End of ZLAQR0 ====
*
*          END

```

— LAPACK zlaqr0 —

```

(let*
  ((ntiny 11) (kexnw 5) (kexsh 6) (wilk1 0.75d0)
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (two 2.0d0))
  (declare (type (f2cl-lib:integer4 11 11) ntiny)
            (type (f2cl-lib:integer4 5 5) kexnw) (type (f2cl-lib:integer4 6 6) kexsh)
            (type (double-float 0.75d0 0.75d0) wilk1) (type (f2cl-lib:complex16) zero)
            (type (f2cl-lib:complex16) one) (type (double-float 2.0d0 2.0d0) two)
            (ignorable ntiny kexnw kexsh wilk1 zero one two))
  (defun zlaqr0 (wantt wantz n ilo ihi h ldh w iloz ihiz z ldz work lwork info)

```

```

(declare (type f2cl-lib:logical wantz wantt)
  (type (f2cl-lib:integer4) info lwork ldz ihiz iloz ldh ihi ilo n)
  (type (array f2cl-lib:complex16 (*)) work z w h))
(f2cl-lib:with-multi-array-data
  ((h f2cl-lib:complex16 h-%data% h-%offset%)
   (w f2cl-lib:complex16 w-%data% w-%offset%)
   (z f2cl-lib:complex16 z-%data% z-%offset%)
   (work f2cl-lib:complex16 work-%data% work-%offset%))
  (labels
    ((cabs1 (cdum) (+ (abs (f2cl-lib:db1e cdum))
                      (abs (f2cl-lib:dimag cdum)))))
  (declare
    (ftype (function (f2cl-lib:complex16)
                     (values double-float &rest t)) cabs1))
  (prog
    ((zdum (make-array 1 :element-type 'f2cl-lib:complex16))
     (jbcmpz (make-array '(2) :element-type 'character
                        :initial-element #\space))
     (sorted nil) (i 0) (inf 0) (it 0) (itmax 0) (k 0) (kacc22 0) (kbot 0)
     (kdu 0) (ks 0) (kt 0) (ktop 0) (ku 0) (kv 0) (kwh 0) (kwt0p 0)
     (kwv 0)
     (ld 0) (ls 0) (lwko1t 0) (ndec 0) (ndfl 0) (nh 0) (nho 0) (nibble 0)
     (nmin 0) (ns 0) (nsmax 0) (nsr 0) (nve 0) (nw 0) (nwmax 0) (nwr 0)
     (nwupbd 0) (s 0.0d0) (aa #C(0.0d0 0.0d0)) (bb #C(0.0d0 0.0d0))
     (cc #C(0.0d0 0.0d0)) (cdum #C(0.0d0 0.0d0)) (dd #C(0.0d0 0.0d0))
     (det #C(0.0d0 0.0d0)) (rtdisc #C(0.0d0 0.0d0)) (swap #C(0.0d0 0.0d0))
     (tr2 #C(0.0d0 0.0d0)))
    (declare (type (array f2cl-lib:complex16 (1)) zdum)
      (type (simple-array character (2)) jbcmpz)
      (type f2cl-lib:logical sorted)
      (type (f2cl-lib:integer4) nwupbd nwr nwmax nw nve nsr nsmax ns
        nmin nibble
        nho nh ndfl ndec lwko1t ls ld kwv kwtop kwh kv ku ktop kt ks
        kdu kbot
        kacc22 k itmax it inf i)
      (type (double-float) s)
      (type (f2cl-lib:complex16) tr2 swap rtdisc det dd cdum cc bb aa))
    (setf info 0)
    (cond
      ((= n 0)
       (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
       (go end_label)))
    (cond
      ((<= n ntiny) (setf lwko1t 1)
       (if (/= lwork -1)
         (multiple-value-bind
           (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
            var-9 var-10
            var-11 var-12)
           (zlahqr wantt wantz n ilo ihi h ldh w iloz ihiz z ldz info)

```

```

(declare
  (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7 var-8
    var-9 var-10
    var-11))
(setf ldh var-6) (setf info var-12))))
(t
  (tagbody (setf info 0)
    (cond
      (wantt (f2cl-lib:fset-string
        (f2cl-lib:fref-string jbcmpz (1 1)) "S"))
      (t (f2cl-lib:fset-string
        (f2cl-lib:fref-string jbcmpz (1 1)) "E"))))
    (cond
      (wantz (f2cl-lib:fset-string
        (f2cl-lib:fref-string jbcmpz (2 2)) "V"))
      (t (f2cl-lib:fset-string
        (f2cl-lib:fref-string jbcmpz (2 2)) "N"))))
    (setf nwr
      (multiple-value-bind
        (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
        (ilaenv 13 "ZLAQR0" jbcmpz n ilo ihi lwork)
        (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
        (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
        (when var-5 (setf ihi var-5))
        (when var-6 (setf lwork var-6)) ret-val))
      (setf nwr (max (the f2cl-lib:integer4 2)
        (the f2cl-lib:integer4 nwr))))
    (setf nwr
      (min (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1)
        (the f2cl-lib:integer4 (truncate (- n 1) 3)) nwr))
    (setf nsr
      (multiple-value-bind
        (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
        (ilaenv 15 "ZLAQR0" jbcmpz n ilo ihi lwork)
        (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
        (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
        (when var-5 (setf ihi var-5))
        (when var-6 (setf lwork var-6)) ret-val))
      (setf nsr
        (min nsr (the f2cl-lib:integer4 (truncate (+ n 6) 9))
          (f2cl-lib:int-sub ihi ilo)))
      (setf nsr
        (max (the f2cl-lib:integer4 2)
          (the f2cl-lib:integer4 (f2cl-lib:int-sub nsr (mod nsr 2))))))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-9
        var-10
        var-11 var-12 var-13 var-14 var-15 var-16 var-17 var-18 var-19
        var-20
        var-21 var-22 var-23 var-24)

```

```

(zlaqr3 wantt wantz n ilo ihi (f2cl-lib:int-add nwr 1) h
  ldh iloz ihiz z
  ldz ls ld w h ldh n h ldh n h ldh work -1)
(declare
  (ignore var-5 var-6 var-10 var-14 var-15 var-18 var-21
    var-23 var-24))
(when var-0 (setf wantt var-0)) (when var-1 (setf wantz var-1))
(when var-2 (setf n var-2)) (when var-3 (setf ilo var-3))
(when var-4 (setf ihi var-4)) (when var-7 (setf ldh var-7))
(when var-8 (setf iloz var-8)) (when var-9 (setf ihiz var-9))
(when var-11 (setf ldz var-11)) (when var-12 (setf ls var-12))
(when var-13 (setf ld var-13)) (when var-16 (setf ldh var-16))
(when var-17 (setf n var-17)) (when var-19 (setf ldh var-19))
(when var-20 (setf n var-20)) (when var-22 (setf ldh var-22))
(setf lwkopt
  (max (the f2cl-lib:integer4 (truncate (* 3 nwr) 2))
    (f2cl-lib:int (f2cl-lib:fref work-%data% (1) ((1 *))
      work-%offset%))))
(cond
  ((= lwork (f2cl-lib:int-sub 1))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (f2cl-lib:dcmplx lwkopt 0))
    (go end_label)))
(setf nmin
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 12 "ZLAQR0" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nmin
  (max (the f2cl-lib:integer4 ntiny) (the f2cl-lib:integer4 nmin)))
(setf nibble
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 14 "ZLAQR0" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nibble
  (max (the f2cl-lib:integer4 0) (the f2cl-lib:integer4 nibble)))
(setf kacc22
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 16 "ZLAQR0" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))

```

```

      (when var-6 (setf lwork var-6)) ret-val))
(setf kacc22
  (max (the f2cl-lib:integer4 0) (the f2cl-lib:integer4 kacc22)))
(setf kacc22
  (min (the f2cl-lib:integer4 2) (the f2cl-lib:integer4 kacc22)))
(setf nwmax
  (min (the f2cl-lib:integer4 (truncate (- n 1) 3))
    (the f2cl-lib:integer4 (truncate lwork 2))))
(setf nw nwmax)
(setf nsmax
  (min (the f2cl-lib:integer4 (truncate (+ n 6) 9))
    (the f2cl-lib:integer4 (truncate (* 2 lwork) 3))))
(setf nsmax (f2cl-lib:int-sub nsmax (mod nsmax 2))) (setf ndfl 1)
(setf itmax
  (f2cl-lib:int-mul
    (max (the f2cl-lib:integer4 30)
      (the f2cl-lib:integer4 (f2cl-lib:int-mul 2 kexsh)))
    (max (the f2cl-lib:integer4 10)
      (the f2cl-lib:integer4
        (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1)))))
(setf kbot ihi)
(f2cl-lib:fdo (it 1 (f2cl-lib:int-add it 1))
  (> it itmax) nil)
  (tagbody
    (if (< kbot ilo) (go label80))
    (f2cl-lib:fdo (k kbot
      (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
      (> k
        (f2cl-lib:int-add ilo 1))
      nil)
    (tagbody
      (if
        (=
          (f2cl-lib:fref h-%data% (k
            (f2cl-lib:int-sub k 1)) ((1 ldh) (1 *))
            h-%offset%)
          zero)
        (go label20))
      label10))
    (setf k ilo) label20 (setf ktop k)
    (setf nh (f2cl-lib:int-add
      (f2cl-lib:int-sub kbot ktop) 1))
    (setf nwupbd
      (min (the f2cl-lib:integer4 nh)
        (the f2cl-lib:integer4 nwmax)))
    (cond
      ((< ndfl kexnw)
        (setf nw
          (min (the f2cl-lib:integer4 nwupbd)
            (the f2cl-lib:integer4 nwr))))

```

```

(t
  (setf nw
    (min (the f2cl-lib:integer4 nwupbd)
      (the f2cl-lib:integer4
        (f2cl-lib:int-mul 2 nw))))))
(cond
  ((< nw nwmax)
    (cond ((>= nw (f2cl-lib:int-add nh
      (f2cl-lib:int-sub 1))) (setf nw nh))
    (t (setf kwtop (f2cl-lib:int-add
      (f2cl-lib:int-sub kbot nw) 1))
      (if
        (>
          (cabs1
            (f2cl-lib:fref h-%data% (kwtop
              (f2cl-lib:int-sub kwtop 1))
            ((1 ldh) (1 *)) h-%offset%))
          (cabs1
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-sub kwtop 1)
                (f2cl-lib:int-sub kwtop 2))
            ((1 ldh) (1 *)) h-%offset%))
            (setf nw (f2cl-lib:int-add nw 1))))))
    (cond ((< ndfl kexnw) (setf ndec -1))
      ((or (>= ndec 0) (>= nw nwupbd))
        (setf ndec (f2cl-lib:int-add ndec 1))
        (if (< (f2cl-lib:int-sub nw ndec) 2) (setf ndec 0))
        (setf nw (f2cl-lib:int-sub nw ndec))))
    (setf kv (f2cl-lib:int-add
      (f2cl-lib:int-sub n nw) 1))
    (setf kt (f2cl-lib:int-add nw 1))
    (setf nho (f2cl-lib:int-add
      (f2cl-lib:int-sub n nw 1 kt) 1))
    (setf kwv (f2cl-lib:int-add nw 2))
    (setf nve
      (f2cl-lib:int-add (f2cl-lib:int-sub n nw kwv) 1))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
        var-7 var-8 var-9 var-10
        var-11 var-12 var-13 var-14 var-15 var-16
        var-17 var-18 var-19 var-20
        var-21 var-22 var-23 var-24)
      (zlaqr3 wantt wantz n ktop kbot nw h ldh iloz
        ihiz z ldz ls ld w
        (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
          (kv 1)
          ((1 ldh) (1 *)) h-%offset%)
        ldh nho
        (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
          (kv kt)

```



```

      ((1 ldh) (1 *)) h-%offset%)
ldh nve
(f2cl-lib:array-slice h-%data% f2cl-lib:complex16
  (kwv 1)
  ((1 ldh) (1 *)) h-%offset%)
ldh work lwork)
(declare (ignore var-6 var-10 var-14 var-15
           var-18 var-21 var-23))
(when var-0 (setf wantt var-0))
(when var-1 (setf wantz var-1))
(when var-2 (setf n var-2))
(when var-3 (setf ktop var-3))
(when var-4 (setf kbot var-4))
(when var-5 (setf nw var-5))
(when var-7 (setf ldh var-7))
(when var-8 (setf iloz var-8))
(when var-9 (setf ihiz var-9))
(when var-11 (setf ldz var-11))
(when var-12 (setf ls var-12))
(when var-13 (setf ld var-13))
(when var-16 (setf ldh var-16))
(when var-17 (setf nho var-17))
(when var-19 (setf ldh var-19))
(when var-20 (setf nve var-20))
(when var-22 (setf ldh var-22))
(when var-24 (setf lwork var-24)))
(setf kbot (f2cl-lib:int-sub kbot ld))
(setf ks (f2cl-lib:int-add
          (f2cl-lib:int-sub kbot ls) 1))
(cond
  ((or (= ld 0)
        (and (<= (f2cl-lib:int-mul 100 ld)
                  (f2cl-lib:int-mul nw nibble))
              (> (f2cl-lib:int-add kbot
                                   (f2cl-lib:int-sub ktop) 1)
                  (min (the f2cl-lib:integer4 nmin)
                        (the f2cl-lib:integer4 nwmax))))))
    (setf ns
      (min (the f2cl-lib:integer4 nsmax)
            (the f2cl-lib:integer4 nsr)
            (the f2cl-lib:integer4
              (max (the f2cl-lib:integer4 2)
                    (the f2cl-lib:integer4
                      (f2cl-lib:int-sub kbot ktop)))))))
    (setf ns (f2cl-lib:int-sub ns (mod ns 2)))
    (cond
      ((= (mod ndfl kexsh) 0)
        (setf ks (f2cl-lib:int-add
                  (f2cl-lib:int-sub kbot ns) 1))
        (f2cl-lib:fdo (i kbot (f2cl-lib:int-add i

```

```

(f2cl-lib:int-sub 2)))
(<> i
  (f2cl-lib:int-add ks 1))
nil)
(tagbody
  (setf (f2cl-lib:fref w-%data% (i)
    ((1 *)) w-%offset%)
    (+ (f2cl-lib:fref h-%data% (i i)
      ((1 ldh) (1 *)) h-%offset%)
      (* wilk1
        (cabs1
          (f2cl-lib:fref h-%data%
            (i (f2cl-lib:int-sub i 1))
            ((1 ldh) (1 *)) h-%offset%))))))
    (setf
      (f2cl-lib:fref w-%data%
        ((f2cl-lib:int-sub i 1)) ((1 *))
        w-%offset%)
      (f2cl-lib:fref w-%data% (i) ((1 *))
        w-%offset%)
      label30)))
(t
  (cond
    ((<= (f2cl-lib:int-add kbot
      (f2cl-lib:int-sub ks) 1)
      (f2cl-lib:f2cl/ ns 2))
      (setf ks (f2cl-lib:int-add
        (f2cl-lib:int-sub kbot ns) 1))
      (setf kt (f2cl-lib:int-add
        (f2cl-lib:int-sub n ns) 1))
      (zlacpy "A" ns ns
        (f2cl-lib:array-slice h-%data%
          f2cl-lib:complex16 (ks ks)
          ((1 ldh) (1 *)) h-%offset%)
        ldh
        (f2cl-lib:array-slice h-%data%
          f2cl-lib:complex16 (kt 1)
          ((1 ldh) (1 *)) h-%offset%)
        ldh)
      (cond
        ((> ns nmin)
          (multiple-value-bind
            (var-0 var-1 var-2 var-3 var-4 var-5
              var-6 var-7 var-8 var-9
              var-10 var-11 var-12 var-13 var-14)
            (zlaqr4 f2cl-lib:%false% f2cl-lib:%false%
              ns 1 ns
                (f2cl-lib:array-slice h-%data%
                  f2cl-lib:complex16 (kt 1)
                  ((1 ldh) (1 *)) h-%offset%)

```

```

ldh
(f2cl-lib:array-slice w-%data%
 f2cl-lib:complex16 (ks) ((1 *))
 w-%offset%)
1 1 zdum 1 work lwork inf)
(declare
(ignore var-0 var-1 var-3 var-5 var-7
 var-8 var-9 var-10 var-11
 var-12))
(when var-2 (setf ns var-2))
(when var-4 (setf ns var-4))
(when var-6 (setf ldh var-6))
(when var-13 (setf lwork var-13))
(when var-14 (setf inf var-14)))
(t
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5
 var-6 var-7 var-8 var-9
 var-10 var-11 var-12)
(zlahqr f2cl-lib:%false% f2cl-lib:%false%
 ns 1 ns
(f2cl-lib:array-slice h-%data%
 f2cl-lib:complex16 (kt 1)
 ((1 ldh) (1 *)) h-%offset%)
ldh
(f2cl-lib:array-slice w-%data%
 f2cl-lib:complex16 (ks) ((1 *))
 w-%offset%)
1 1 zdum 1 inf)
(declare
(ignore var-0 var-1 var-2 var-3 var-4
 var-5 var-7 var-8 var-9
 var-10 var-11))
(setf ldh var-6) (setf inf var-12)))
(setf ks (f2cl-lib:int-add ks inf))
(cond
(>= ks kbot)
(setf s
(+
(cabs1
(f2cl-lib:fref h-%data%
((f2cl-lib:int-sub kbot 1)
(f2cl-lib:int-sub kbot 1))
((1 ldh) (1 *)) h-%offset%))
(cabs1
(f2cl-lib:fref h-%data% (kbot
(f2cl-lib:int-sub kbot 1))
((1 ldh) (1 *)) h-%offset%))
(cabs1
(f2cl-lib:fref h-%data%

```

```

        ((f2cl-lib:int-sub kbot 1) kbot)
        ((1 ldh) (1 *)) h-%offset%))
(cabs1
 (f2cl-lib:fref h-%data% (kbot kbot)
  ((1 ldh) (1 *))
  h-%offset%)))
(setf aa
 (/
 (f2cl-lib:fref h-%data%
  ((f2cl-lib:int-sub kbot 1)
   (f2cl-lib:int-sub kbot 1))
  ((1 ldh) (1 *)) h-%offset%
 s))
(setf cc
 (/
 (f2cl-lib:fref h-%data% (kbot
  (f2cl-lib:int-sub kbot 1))
  ((1 ldh) (1 *)) h-%offset%
 s))
(setf bb
 (/
 (f2cl-lib:fref h-%data%
  ((f2cl-lib:int-sub kbot 1) kbot)
  ((1 ldh) (1 *)) h-%offset%
 s))
(setf dd
 (/
 (f2cl-lib:fref h-%data% (kbot kbot)
  ((1 ldh) (1 *)) h-%offset%
 s))
(setf tr2 (/ (+ aa dd) two))
(setf det (- (* (- aa tr2) (- dd tr2))
  (* bb cc)))
(setf rtdisc (f2cl-lib:fsqrt (- det)))
(setf
 (f2cl-lib:fref w-%data%
  ((f2cl-lib:int-sub kbot 1) ((1 *))
  w-%offset%)
 (* (+ tr2 rtdisc) s))
(setf (f2cl-lib:fref w-%data% (kbot)
  ((1 *)) w-%offset%)
 (* (- tr2 rtdisc) s))
(setf ks (f2cl-lib:int-sub kbot 1))))))
(cond
 (> (f2cl-lib:int-add kbot
  (f2cl-lib:int-sub ks) 1) ns)
(tagbody (setf sorted f2cl-lib:%false%)
 (f2cl-lib:fdo (k kbot
  (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
  (>

```

```

        k (f2cl-lib:int-add ks 1))
      nil)
    (tagbody (if sorted (go label60))
      (setf sorted f2cl-lib:%true%)
      (f2cl-lib:fdo (i ks
        (f2cl-lib:int-add i 1))
        ((> i
          (f2cl-lib:int-add k
            (f2cl-lib:int-sub 1)))
          nil)
      (tagbody
        (cond
          ((< (cabs1 (f2cl-lib:fref w (i)
            ((1 *))))
            (cabs1 (f2cl-lib:fref w
              ((f2cl-lib:int-add i 1))
              ((1 *))))))
            (setf sorted f2cl-lib:%false%)
            (setf swap
              (f2cl-lib:fref w-%data% (i)
                ((1 *)) w-%offset%))
            (setf
              (f2cl-lib:fref w-%data% (i)
                ((1 *)) w-%offset%)
              (f2cl-lib:fref w-%data%
                ((f2cl-lib:int-add i 1))
                ((1 *))
                w-%offset%))
            (setf
              (f2cl-lib:fref w-%data%
                ((f2cl-lib:int-add i 1))
                ((1 *))
                w-%offset%)
              swap)))
          label40))
        label50))
      label60))))))
    (cond
      ((= (f2cl-lib:int-add kbot
        (f2cl-lib:int-sub ks) 1) 2)
      (cond
        ((<
          (cabs1
            (+ (f2cl-lib:fref w (kbot) ((1 *)))
              (- (f2cl-lib:fref h (kbot kbot) ((1 ldh)
                (1 *))))))
          (cabs1
            (+
              (f2cl-lib:fref w ((f2cl-lib:int-add kbot
                (f2cl-lib:int-sub 1)))

```

```

      ((1 *)))
      (- (f2cl-lib:fref h (kbot kbot) ((1 ldh)
      (1 *))))))
(setf
  (f2cl-lib:fref w-%data%
    ((f2cl-lib:int-sub kbot 1)) ((1 *))
    w-%offset%)
  (f2cl-lib:fref w-%data% (kbot) ((1 *))
    w-%offset%))
(t
  (setf (f2cl-lib:fref w-%data% (kbot) ((1 *))
    w-%offset%)
    (f2cl-lib:fref w-%data%
      ((f2cl-lib:int-sub kbot 1)) ((1 *))
      w-%offset%))))))
(setf ns
  (min (the f2cl-lib:integer4 ns)
    (the f2cl-lib:integer4
      (f2cl-lib:int-add
        (f2cl-lib:int-sub kbot ks) 1))))
(setf ns (f2cl-lib:int-sub ns (mod ns 2)))
(setf ks (f2cl-lib:int-add
  (f2cl-lib:int-sub kbot ns) 1))
(setf kdu (f2cl-lib:int-sub
  (f2cl-lib:int-mul 3 ns) 3))
(setf ku (f2cl-lib:int-add
  (f2cl-lib:int-sub n kdu) 1))
(setf kwh (f2cl-lib:int-add kdu 1))
(setf nho
  (f2cl-lib:int-add
    (f2cl-lib:int-sub (f2cl-lib:int-add
      (f2cl-lib:int-sub n kdu) 1) 4
      (f2cl-lib:int-add kdu 1))
    1))
(setf kwv (f2cl-lib:int-add kdu 4))
(setf nve (f2cl-lib:int-add
  (f2cl-lib:int-sub n kdu kwv) 1))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10
    var-11 var-12 var-13 var-14 var-15 var-16
    var-17 var-18 var-19 var-20
    var-21 var-22 var-23)
  (zlaqr5 wantt wantz kacc22 n ktop kbot ns
    (f2cl-lib:array-slice w-%data%
      f2cl-lib:complex16 (ks) ((1 *))
      w-%offset%)
    h ldh iloz ihiz z ldz work 3
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (ku 1)

```

```

      ((1 ldh) (1 *)) h-%offset%)
    ldh nve
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (kwv 1)
      ((1 ldh) (1 *)) h-%offset%)
    ldh nho
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (ku kwh)
      ((1 ldh) (1 *)) h-%offset%)
    ldh)
  (declare
    (ignore var-7 var-8 var-12 var-14 var-15
      var-16 var-19 var-22))
  (when var-0 (setf wantt var-0))
  (when var-1 (setf wantz var-1))
  (when var-2 (setf kacc22 var-2))
  (when var-3 (setf n var-3))
  (when var-4 (setf ktop var-4))
  (when var-5 (setf kbot var-5))
  (when var-6 (setf ns var-6))
  (when var-9 (setf ldh var-9))
  (when var-10 (setf iloz var-10))
  (when var-11 (setf ihiz var-11))
  (when var-13 (setf ldz var-13))
  (when var-17 (setf ldh var-17))
  (when var-18 (setf nve var-18))
  (when var-20 (setf ldh var-20))
  (when var-21 (setf nho var-21))
  (when var-23 (setf ldh var-23))))))
  (cond ((> ld 0) (setf ndfl 1))
    (t (setf ndfl (f2cl-lib:int-add ndfl 1))))
  label70))
  (setf info kbot) label80)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (f2cl-lib:dcmplx lwkopt 0))
end_label
(return
  (values wantt wantz n ilo ihi nil ldh nil iloz ihiz nil ldz nil lwork
    info))))))

```

zlaqr1 LAPACK

— zlaqr1.input —

)set break resume

```

)sys rm -f zlaqr1.output
)spool zlaqr1.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlaqr1.help —

```

=====
zlaqr1 examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```

SUBROUTINE ZLAQR1( N, H, LDH, S1, S2, V )

```

```

.. Scalar Arguments ..

```

```

COMPLEX*16      S1, S2

```

```

INTEGER        LDH, N

```

```

..

```

```

.. Array Arguments ..

```

```

COMPLEX*16      H( LDH, * ), V( * )

```

```

..

```

Purpose:

=====

Given a 2-by-2 or 3-by-3 matrix H, ZLAQR1 sets v to a scalar multiple of the first column of the product

$$(*) \quad K = (H - s_1 I)(H - s_2 I)$$

scaling to avoid overflows and most underflows.

This is useful for starting double implicit shift bulges

in the QR algorithm.

Arguments:
=====

[in] N

N is integer
Order of the matrix H. N must be either 2 or 3.

[in] H

H is COMPLEX*16 array of dimension (LDH,N)
The 2-by-2 or 3-by-3 matrix H in (*).

[in] LDH

LDH is integer
The leading dimension of H as declared in
the calling procedure. LDH.GE.N

[in] S1

S1 is COMPLEX*16

[in] S2

S2 is COMPLEX*16

S1 and S2 are the shifts defining K in (*) above.

[out] V

V is COMPLEX*16 array of dimension N
A scalar multiple of the first column of the
matrix K in (*).

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Contributors:
=====

Karen Braman and Ralph Byers, Department of Mathematics,
University of Kansas, USA

— zlaqr1.f —

```
* =====
*      SUBROUTINE ZLAQR1( N, H, LDH, S1, S2, V )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,      --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      COMPLEX*16      S1, S2
*      INTEGER         LDH, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16      H( LDH, * ), V( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
*      COMPLEX*16      ZERO
*      PARAMETER       ( ZERO = ( 0.0d0, 0.0d0 ) )
*      DOUBLE PRECISION RZERO
*      PARAMETER       ( RZERO = 0.0d0 )
*
*      ..
*      .. Local Scalars ..
*      COMPLEX*16      CDUM, H21S, H31S
*      DOUBLE PRECISION S
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC       ABS, DBLE, DIMAG
*
*      ..
*      .. Statement Functions ..
*      DOUBLE PRECISION CABS1
*
*      ..
*      .. Statement Function definitions ..
*      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
*      ..
*      .. Executable Statements ..
*      IF( N.EQ.2 ) THEN
*          S = CABS1( H( 1, 1 )-S2 ) + CABS1( H( 2, 1 ) )
*          IF( S.EQ.RZERO ) THEN
```

```

      V( 1 ) = ZERO
      V( 2 ) = ZERO
    ELSE
      H21S = H( 2, 1 ) / S
      V( 1 ) = H21S*H( 1, 2 ) + ( H( 1, 1 )-S1 )*
$      ( ( H( 1, 1 )-S2 ) / S )
      V( 2 ) = H21S*( H( 1, 1 )+H( 2, 2 )-S1-S2 )
    END IF
  ELSE
    S = CABS1( H( 1, 1 )-S2 ) + CABS1( H( 2, 1 ) ) +
$    CABS1( H( 3, 1 ) )
    IF( S.EQ.ZERO ) THEN
      V( 1 ) = ZERO
      V( 2 ) = ZERO
      V( 3 ) = ZERO
    ELSE
      H21S = H( 2, 1 ) / S
      H31S = H( 3, 1 ) / S
      V( 1 ) = ( H( 1, 1 )-S1 )*( ( H( 1, 1 )-S2 ) / S ) +
$      H( 1, 2 )*H21S + H( 1, 3 )*H31S
      V( 2 ) = H21S*( H( 1, 1 )+H( 2, 2 )-S1-S2 ) + H( 2, 3 )*H31S
      V( 3 ) = H31S*( H( 1, 1 )+H( 3, 3 )-S1-S2 ) + H21S*H( 3, 2 )
    END IF
  END IF
END

```

— LAPACK zlaqr1 —

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (rzero 0.0d0))
  (declare (type (f2cl-lib:complex16) zero)
            (type (double-float 0.0d0 0.0d0) rzero) (ignorable zero rzero))
  (defun zlaqr1 (n h ldh s1 s2 v)
    (declare (type (f2cl-lib:integer4) ldh n)
              (type (array f2cl-lib:complex16 (*)) v h) (type (f2cl-lib:complex16) s2 s1))
    (f2cl-lib:with-multi-array-data
      ((h f2cl-lib:complex16 h-%data% h-%offset%)
       (v f2cl-lib:complex16 v-%data% v-%offset%))
      (labels
        ((cabs1 (cdum) (+ (abs (f2cl-lib:dbple cdum))
                          (abs (f2cl-lib:dimag cdum)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                           (values double-float &rest t)) cabs1))
        (prog

```

```

((s 0.0d0) (cdum #C(0.0d0 0.0d0)) (h21s #C(0.0d0 0.0d0))
 (h31s #C(0.0d0 0.0d0)))
(declare (type (double-float) s)
         (type (f2cl-lib:complex16) h31s h21s cdum))
(cond
  ((= n 2)
   (setf s
    (+
      (cabs1 (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *))
                  h-%offset%) s2))
      (cabs1 (f2cl-lib:fref h-%data% (2 1) ((1 ldh) (1 *))
              h-%offset%))))))
  (cond
    ((= s rzero)
     (setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%) zero)
     (setf (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%) zero))
    (t
     (setf h21s
      (/ (f2cl-lib:fref h-%data% (2 1) ((1 ldh) (1 *)) h-%offset%) s))
     (setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
      (+ (* h21s (f2cl-lib:fref h-%data% (1 2) ((1 ldh) (1 *))
                          h-%offset%))
        (* (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *))
              h-%offset%) s1)
          (/ (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *))
              h-%offset%) s2)
            s))))))
     (setf (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%)
      (* h21s
        (-
          (+ (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *)) h-%offset%)
            (f2cl-lib:fref h-%data% (2 2) ((1 ldh) (1 *)) h-%offset%)
            s1 s2))))))
    (t
     (setf s
      (+
        (cabs1
          (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *)) h-%offset%) s2))
        (cabs1 (f2cl-lib:fref h-%data% (2 1) ((1 ldh) (1 *)) h-%offset%))
        (cabs1
          (f2cl-lib:fref h-%data% (3 1) ((1 ldh) (1 *)) h-%offset%))))))
     (cond
       ((= s zero)
        (setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%) zero)
        (setf (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%) zero)
        (setf (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%) zero))
       (t
        (setf h21s
         (/ (f2cl-lib:fref h-%data% (2 1) ((1 ldh) (1 *)) h-%offset%) s))
        (setf h31s

```

```

(/ (f2cl-lib:fref h-%data% (3 1) ((1 ldh) (1 *)) h-%offset%) s))
(setf (f2cl-lib:fref v-%data% (1) ((1 *)) v-%offset%)
(+
(* (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *))
      h-%offset%) s1)
(/ (- (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *))
      h-%offset%) s2)
s))
(* (f2cl-lib:fref h-%data% (1 2) ((1 ldh) (1 *)) h-%offset%)
h21s)
(* (f2cl-lib:fref h-%data% (1 3) ((1 ldh) (1 *)) h-%offset%)
h31s)))
(setf (f2cl-lib:fref v-%data% (2) ((1 *)) v-%offset%)
(+
(* h21s
(-
(+ (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *)) h-%offset%)
(f2cl-lib:fref h-%data% (2 2) ((1 ldh) (1 *)) h-%offset%)
s1 s2))
(* (f2cl-lib:fref h-%data% (2 3) ((1 ldh) (1 *)) h-%offset%)
h31s)))
(setf (f2cl-lib:fref v-%data% (3) ((1 *)) v-%offset%)
(+
(* h31s
(-
(+ (f2cl-lib:fref h-%data% (1 1) ((1 ldh) (1 *)) h-%offset%)
(f2cl-lib:fref h-%data% (3 3) ((1 ldh) (1 *)) h-%offset%)
s1 s2))
(* h21s
(f2cl-lib:fref h-%data% (3 2) ((1 ldh) (1 *)) h-%offset%))))))
end_label (return (values nil nil nil nil nil nil))))))

```

zlaqr2 LAPACK

— zlaqr2.input —

```

)set break resume
)sys rm -f zlaqr2.output
)spool zlaqr2.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlaqr2.help —

```
=====
zlaqr2 examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLAQR2( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
                  IHIZ, Z, LDZ, NS, ND, SH, V, LDV, NH, T, LDT,
                  NV, WV, LDWV, WORK, LWORK )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHIZ, ILOZ, KBOT, KTOP, LDH, LDT, LDV, LDWV,
$               LDZ, LWORK, N, ND, NH, NS, NV, NW
LOGICAL          WANTT, WANTZ
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       H( LDH, * ), SH( * ), T( LDT, * ), V( LDV, * ),
$               WORK( * ), WV( LDWV, * ), Z( LDZ, * )
```

```
..
```

Purpose:

```
=====
```

ZLAQR2 is identical to ZLAQR3 except that it avoids recursion by calling ZLAHQR instead of ZLAQR4.

Aggressive early deflation:

ZLAQR2 accepts as input an upper Hessenberg matrix H and performs an unitary similarity transformation designed to detect and deflate fully converged eigenvalues from a trailing principal submatrix. On output H has been overwritten by a new Hessenberg matrix that is a perturbation of an unitary similarity transformation of H. It is to be hoped that the final version of H has many zero subdiagonal entries.

Arguments:

=====

[in] WANTT

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine).
If .FALSE., then only enough of H is updated to preserve the eigenvalues.

[in] WANTZ

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine).
If .FALSE., then Z is not referenced.

[in] N

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

[in] KTOP

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0.
KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

[in] KBOT

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1,KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

[in] NW

NW is INTEGER

Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)

On input the initial N-by-N section of H stores the

Hessenberg matrix undergoing aggressive early deflation.
On output H has been transformed by a unitary
similarity transformation, perturbed, and the returned
to Hessenberg form that (it is to be hoped) has some
zero subdiagonal entries.

[in] LDH

LDH is integer
Leading dimension of H just as declared in the calling
subroutine. N .LE. LDH

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER
Specify the rows of Z to which transformations must be
applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,N)
IF WANTZ is .TRUE., then on output, the unitary
similarity transformation mentioned above has been
accumulated into Z(ILOZ:IHIZ,ILO:IHI) from the right.
If WANTZ is .FALSE., then Z is unreferenced.

[in] LDZ

LDZ is integer
The leading dimension of Z just as declared in the
calling subroutine. 1 .LE. LDZ.

[out] NS

NS is integer
The number of unconverged (ie approximate) eigenvalues
returned in SR and SI that may be used as shifts by the
calling subroutine.

[out] ND

ND is integer
The number of converged eigenvalues uncovered by this
subroutine.

[out] SH

SH is COMPLEX*16 array, dimension KBOT
On output, approximate eigenvalues that may
be used for shifts are stored in SH(KBOT-ND-NS+1)
through SR(KBOT-ND). Converged eigenvalues are
stored in SH(KBOT-ND+1) through SH(KBOT).

[out] V

V is COMPLEX*16 array, dimension (LDV,NW)
An NW-by-NW work array.

[in] LDV

LDV is integer scalar
The leading dimension of V just as declared in the
calling subroutine. NW .LE. LDV

[in] NH

NH is integer scalar
The number of columns of T. NH.GE.NW.

[out] T

T is COMPLEX*16 array, dimension (LDT,NW)

[in] LDT

LDT is integer
The leading dimension of T just as declared in the
calling subroutine. NW .LE. LDT

[in] NV

NV is integer
The number of rows of work array WV available for
workspace. NV.GE.NW.

[out] WV

WV is COMPLEX*16 array, dimension (LDWV,NW)

[in] LDWV

LDWV is integer
The leading dimension of W just as declared in the
calling subroutine. NW .LE. LDV

[out] WORK

WORK is COMPLEX*16 array, dimension LWORK.
 On exit, WORK(1) is set to an estimate of the optimal value
 of LWORK for the given values of N, NW, KTOP and KBOT.

[in] LWORK

LWORK is integer
 The dimension of the work array WORK. LWORK = 2*NW
 suffices, but greater efficiency may result from larger
 values of LWORK.

If LWORK = -1, then a workspace query is assumed; ZLAQR2
 only estimates the optimal workspace size for the given
 values of N, NW, KTOP and KBOT. The estimate is returned
 in WORK(1). No error message related to LWORK is issued
 by XERBLA. Neither H nor Z are accessed.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Contributors:

=====

Karen Braman and Ralph Byers, Department of Mathematics,
 University of Kansas, USA

— zlaqr2.f —

```
* =====
*      SUBROUTINE ZLAQR2( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
*      $                  IHIZ, Z, LDZ, NS, ND, SH, V, LDV, NH, T, LDT,
*      $                  NV, WV, LDWV, WORK, LWORK )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
```

```

*      .. Scalar Arguments ..
      INTEGER      IHIZ, ILOZ, KBOT, KTOP, LDH, LDT, LDV, LDWV,
$               LDZ, LWORK, N, ND, NH, NS, NV, NW
      LOGICAL      WANTT, WANTZ
*
*      ..
*      .. Array Arguments ..
      COMPLEX*16    H( LDH, * ), SH( * ), T( LDT, * ), V( LDV, * ),
$               WORK( * ), WV( LDWV, * ), Z( LDZ, * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
      COMPLEX*16    ZERO, ONE
      PARAMETER      ( ZERO = ( 0.0d0, 0.0d0 ),
$               ONE = ( 1.0d0, 0.0d0 ) )
      DOUBLE PRECISION RZERO, RONE
      PARAMETER      ( RZERO = 0.0d0, RONE = 1.0d0 )
*
*      ..
*      .. Local Scalars ..
      COMPLEX*16    BETA, CDUM, S, TAU
      DOUBLE PRECISION FOO, SAFMAX, SAFMIN, SMLNUM, ULP
      INTEGER      I, IFST, ILST, INFO, INFQR, J, JW, KCOL, KLN,
$               KNT, KROW, KWTOP, LTOP, LWK1, LWK2, LWKOPT
*
*      ..
*      .. External Functions ..
      DOUBLE PRECISION DLAMCH
      EXTERNAL      DLAMCH
*
*      ..
*      .. External Subroutines ..
      EXTERNAL      DLABAD, ZCOPY, ZGHERD, ZGEMM, ZLACPY, ZLAHQR,
$               ZLARF, ZLARFG, ZLASET, ZTREXC, ZUNMHR
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC      ABS, DBLE, DCMPLX, DCONJG, DIMAG, INT, MAX, MIN
*
*      ..
*      .. Statement Functions ..
      DOUBLE PRECISION CABS1
*
*      ..
*      .. Statement Function definitions ..
      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
*      ..
*      .. Executable Statements ..
*
*      ==== Estimate optimal workspace. ====
*
      JW = MIN( NW, KBOT-KTOP+1 )
      IF( JW.LE.2 ) THEN
          LWKOPT = 1
      ELSE

```

```

*
*      ==== Workspace query call to ZGEHRD ====
*
      CALL ZGEHRD( JW, 1, JW-1, T, LDT, WORK, WORK, -1, INFO )
      LWK1 = INT( WORK( 1 ) )
*
*      ==== Workspace query call to ZUNMHR ====
*
      CALL ZUNMHR( 'R', 'N', JW, JW, 1, JW-1, T, LDT, WORK, V, LDV,
$           WORK, -1, INFO )
      LWK2 = INT( WORK( 1 ) )
*
*      ==== Optimal workspace ====
*
      LWKOPT = JW + MAX( LWK1, LWK2 )
END IF
*
*      ==== Quick return in case of workspace query. ====
*
      IF( LWORK.EQ.-1 ) THEN
        WORK( 1 ) = DCMLPX( LWKOPT, 0 )
        RETURN
      END IF
*
*      ==== Nothing to do ...
*      ... for an empty active block ... ====
      NS = 0
      ND = 0
      WORK( 1 ) = ONE
      IF( KTOP.GT.KBOT )
$    RETURN
*      ... nor for an empty deflation window. ====
      IF( NW.LT.1 )
$    RETURN
*
*      ==== Machine constants ====
*
      SAFMIN = DLAMCH( 'SAFE MINIMUM' )
      SAFMAX = RONE / SAFMIN
      CALL DLABAD( SAFMIN, SAFMAX )
      ULP = DLAMCH( 'PRECISION' )
      SMLNUM = SAFMIN*( DBLE( N ) / ULP )
*
*      ==== Setup deflation window ====
*
      JW = MIN( NW, KBOT-KTOP+1 )
      KWTOP = KBOT - JW + 1
      IF( KWTOP.EQ.KTOP ) THEN
        S = ZERO
      ELSE

```

```

        S = H( KWTOP, KWTOP-1 )
      END IF
*
      IF( KBOT.EQ.KWTOP ) THEN
*
*       ==== 1-by-1 deflation window: not much to do ====
*
        SH( KWTOP ) = H( KWTOP, KWTOP )
        NS = 1
        ND = 0
        IF( CABS1( S ).LE.MAX( SMLNUM, ULP*CABS1( H( KWTOP,
$          KWTOP ) ) ) ) THEN
          NS = 0
          ND = 1
          IF( KWTOP.GT.KTOP )
$            H( KWTOP, KWTOP-1 ) = ZERO
        END IF
        WORK( 1 ) = ONE
        RETURN
      END IF
*
*     ==== Convert to spike-triangular form. (In case of a
*     .   rare QR failure, this routine continues to do
*     .   aggressive early deflation using that part of
*     .   the deflation window that converged using INFQR
*     .   here and there to keep track.) ====
*
      CALL ZLACPY( 'U', JW, JW, H( KWTOP, KWTOP ), LDH, T, LDT )
      CALL ZCOPY( JW-1, H( KWTOP+1, KWTOP ), LDH+1, T( 2, 1 ), LDT+1 )
*
      CALL ZLASET( 'A', JW, JW, ZERO, ONE, V, LDV )
      CALL ZLAHQR( .true., .true., JW, 1, JW, T, LDT, SH( KWTOP ), 1,
$        JW, V, LDV, INFQR )
*
*     ==== Deflation detection loop ====
*
      NS = JW
      ILST = INFQR + 1
      DO 10 KNT = INFQR + 1, JW
*
*       ==== Small spike tip deflation test ====
*
        FOO = CABS1( T( NS, NS ) )
        IF( FOO.EQ.RZERO )
$          FOO = CABS1( S )
        IF( CABS1( S )*CABS1( V( 1, NS ) ).LE.MAX( SMLNUM, ULP*FOO ) )
$          THEN
*
*         ==== One more converged eigenvalue ====
*

```

```

        NS = NS - 1
    ELSE
*
*      ==== One undeflatable eigenvalue.  Move it up out of the
*      .      way.  (ZTREXC can not fail in this case.) ====
*
        IFST = NS
        CALL ZTREXC( 'V', JW, T, LDT, V, LDV, IFST, ILST, INFO )
        ILST = ILST + 1
    END IF
10 CONTINUE
*
*      ==== Return to Hessenberg form ====
*
        IF( NS.EQ.0 )
$      S = ZERO
*
        IF( NS.LT.JW ) THEN
*
*      ==== sorting the diagonal of T improves accuracy for
*      .      graded matrices.  ====
*
        DO 30 I = INFQR + 1, NS
            IFST = I
            DO 20 J = I + 1, NS
                IF( CABS1( T( J, J ) ).GT.CABS1( T( IFST, IFST ) ) )
$                IFST = J
20            CONTINUE
            ILST = I
            IF( IFST.NE.ILST )
$                CALL ZTREXC( 'V', JW, T, LDT, V, LDV, IFST, ILST, INFO )
30        CONTINUE
        END IF
*
*      ==== Restore shift/eigenvalue array from T ====
*
        DO 40 I = INFQR + 1, JW
            SH( KWTOP+I-1 ) = T( I, I )
40    CONTINUE
*
*
        IF( NS.LT.JW .OR. S.EQ.ZERO ) THEN
            IF( NS.GT.1 .AND. S.NE.ZERO ) THEN
*
*      ==== Reflect spike back into lower triangle ====
*
                CALL ZCOPY( NS, V, LDV, WORK, 1 )
                DO 50 I = 1, NS
                    WORK( I ) = DCONJG( WORK( I ) )
50            CONTINUE

```

```

      BETA = WORK( 1 )
      CALL ZLARFG( NS, BETA, WORK( 2 ), 1, TAU )
      WORK( 1 ) = ONE
*
      CALL ZLASET( 'L', JW-2, JW-2, ZERO, ZERO, T( 3, 1 ), LDT )
*
      CALL ZLARF( 'L', NS, JW, WORK, 1, DCONJG( TAU ), T, LDT,
$           WORK( JW+1 ) )
      CALL ZLARF( 'R', NS, NS, WORK, 1, TAU, T, LDT,
$           WORK( JW+1 ) )
      CALL ZLARF( 'R', JW, NS, WORK, 1, TAU, V, LDV,
$           WORK( JW+1 ) )
*
      CALL ZGHRD( JW, 1, NS, T, LDT, WORK, WORK( JW+1 ),
$           LWORK-JW, INFO )
      END IF
*
*      ==== Copy updated reduced window into place ====
*
      IF( KWTOP.GT.1 )
$       H( KWTOP, KWTOP-1 ) = S*DCONJG( V( 1, 1 ) )
      CALL ZLACPY( 'U', JW, JW, T, LDT, H( KWTOP, KWTOP ), LDH )
      CALL ZCOPY( JW-1, T( 2, 1 ), LDT+1, H( KWTOP+1, KWTOP ),
$           LDH+1 )
*
*      ==== Accumulate orthogonal matrix in order update
*      .      H and Z, if requested.  ====
*
      IF( NS.GT.1 .AND. S.NE.ZERO )
$       CALL ZUNMHR( 'R', 'N', JW, NS, 1, NS, T, LDT, WORK, V, LDV,
$           WORK( JW+1 ), LWORK-JW, INFO )
*
*      ==== Update vertical slab in H ====
*
      IF( WANTT ) THEN
        LTOP = 1
      ELSE
        LTOP = KTOP
      END IF
      DO 60 KROW = LTOP, KWTOP - 1, NV
        KLN = MIN( NV, KWTOP-KROW )
        CALL ZGEMM( 'N', 'N', KLN, JW, JW, ONE, H( KROW, KWTOP ),
$           LDH, V, LDV, ZERO, WV, LDWV )
        CALL ZLACPY( 'A', KLN, JW, WV, LDWV, H( KROW, KWTOP ), LDH )
60    CONTINUE
*
*      ==== Update horizontal slab in H ====
*
      IF( WANTT ) THEN
        DO 70 KCOL = KBOT + 1, N, NH

```

```

        KLN = MIN( NH, N-KCOL+1 )
        CALL ZGEMM( 'C', 'N', JW, KLN, JW, ONE, V, LDV,
$           H( KWTOP, KCOL ), LDH, ZERO, T, LDT )
        CALL ZLACPY( 'A', JW, KLN, T, LDT, H( KWTOP, KCOL ),
$           LDH )
70      CONTINUE
      END IF

*
*      ==== Update vertical slab in Z ====
*
      IF( WANTZ ) THEN
        DO 80 KROW = ILOZ, IHIZ, NV
          KLN = MIN( NV, IHIZ-KROW+1 )
          CALL ZGEMM( 'N', 'N', KLN, JW, JW, ONE, Z( KROW, KWTOP ),
$             LDZ, V, LDV, ZERO, WV, LDWV )
          CALL ZLACPY( 'A', KLN, JW, WV, LDWV, Z( KROW, KWTOP ),
$             LDZ )
80      CONTINUE
      END IF
    END IF

*
*      ==== Return the number of deflations ... ====
*
    ND = JW - NS

*
*      ==== ... and the number of shifts. (Subtracting
*      .   INFQR from the spike length takes care
*      .   of the case of a rare QR failure while
*      .   calculating eigenvalues of the deflation
*      .   window.) ====
*
    NS = NS - INFQR

*
*      ==== Return optimal workspace. ====
*
    WORK( 1 ) = DCMPLX( LWKOPT, 0 )

*
*      ==== End of ZLAQR2 ====
*
  END

```

— LAPACK zlaqr2 —

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (rzero 0.0d0)

```



```

(rone 1.0d0))
(declare (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
  (type (double-float 0.0d0 0.0d0) rzero)
  (type (double-float 1.0d0 1.0d0) rone) (ignorable zero one rzero rone))
(defun zlaqr2
  (wantt wantz n ktop kbot nw h ldh iloz ihiz z ldz ns nd sh v ldv nh t$ ldt nv
    wv ldwv work lwork)
  (declare (type f2cl-lib:logical wantt wantz)
    (type (f2cl-lib:integer4) lwork ldwv nv ldt nh ldv nd ns ldz ihiz iloz ldh
      nw kbot ktop n)
    (type (array f2cl-lib:complex16 (*)) work wv t$ v sh z h))
  (f2cl-lib:with-multi-array-data
    ((h f2cl-lib:complex16 h-%data% h-%offset%)
     (z f2cl-lib:complex16 z-%data% z-%offset%)
     (sh f2cl-lib:complex16 sh-%data% sh-%offset%)
     (v f2cl-lib:complex16 v-%data% v-%offset%)
     (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
     (wv f2cl-lib:complex16 wv-%data% wv-%offset%)
     (work f2cl-lib:complex16 work-%data% work-%offset%))
    (labels
      ((cabs1 (cdum)
        (+ (abs (f2cl-lib:dbble cdum)) (abs (f2cl-lib:dimag cdum)))))
      (declare
        (ftype (function (f2cl-lib:complex16)
          (values double-float &rest t)) cabs1))
      (prog
        ((i 0) (ifst 0) (ilst 0) (info 0) (infqr 0) (j 0) (jw 0)
          (kcol 0) (kln 0)
          (knt 0) (krow 0) (kwtop 0) (ltop 0) (lw1 0) (lw2 0)
          (lwko 0) (foo 0.0d0)
          (safmax 0.0d0) (safmin 0.0d0) (smlnum 0.0d0) (ulp 0.0d0)
          (beta #C(0.0d0 0.0d0)) (cdum #C(0.0d0 0.0d0)) (s #C(0.0d0 0.0d0))
          (tau #C(0.0d0 0.0d0)) (dconjg$ 0.0))
        (declare
          (type (f2cl-lib:integer4) lwko lw2 lw1 ltop kwtop
            krow knt kln kcol jw j
            infqr info ilst ifst i)
          (type (double-float) ulp smlnum safmin safmax foo)
          (type (f2cl-lib:complex16) tau s cdum beta)
          (type (single-float) dconjg$))
        (setf jw
          (min (the f2cl-lib:integer4 nw)
            (the f2cl-lib:integer4
              (f2cl-lib:int-add (f2cl-lib:int-sub kbot ktop) 1))))
        (cond ((<= jw 2) (setf lwko 1))
          (t
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
              (zgehrd jw 1 (f2cl-lib:int-sub jw 1) t$ ldt work work -1 info)
              (declare (ignore var-1 var-2 var-3 var-5 var-6 var-7)))

```

```

(setf jw var-0)
(setf ldt var-4) (setf info var-8))
(setf lwk1
  (f2cl-lib:int
    (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10 var-11
   var-12 var-13)
  (zunmhr "R" "N" jw jw 1
    (f2cl-lib:int-sub jw 1) t$ ldt work v ldv work -1
    info)
  (declare (ignore var-0 var-1 var-4 var-5 var-6 var-8 var-9
    var-11 var-12))
  (when var-2 (setf jw var-2)) (when var-3 (setf jw var-3))
  (when var-7 (setf ldt var-7)) (when var-10 (setf ldv var-10))
  (when var-13 (setf info var-13)))
(setf lwk2
  (f2cl-lib:int (f2cl-lib:fref work-%data% (1) ((1 *))
    work-%offset%)))
(setf lwkopt
  (f2cl-lib:int-add jw
    (max (the f2cl-lib:integer4 lwk1)
      (the f2cl-lib:integer4 lwk2)))))
(cond
  ((= lwork (f2cl-lib:int-sub 1))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (f2cl-lib:dcmplx lwkopt 0))
    (go end_label)))
(setf ns 0) (setf nd 0)
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
(if (> ktop kbot) (go end_label)) (if (< nw 1) (go end_label))
(setf safmin (dlamch "SAFE MINIMUM")) (setf safmax (/ rone safmin))
(multiple-value-bind (var-0 var-1) (dlabad safmin safmax)
  (declare (ignore))
  (when var-0 (setf safmin var-0)) (when var-1 (setf safmax var-1)))
(setf ulp (dlamch "PRECISION"))
(setf smlnum (* safmin (/ (f2cl-lib:db1e n) ulp)))
(setf jw
  (min (the f2cl-lib:integer4 nw)
    (the f2cl-lib:integer4
      (f2cl-lib:int-add (f2cl-lib:int-sub kbot ktop) 1))))
(setf kwtop (f2cl-lib:int-add (f2cl-lib:int-sub kbot jw) 1))
(cond ((= kwtop ktop) (setf s zero))
  (t
    (setf s
      (f2cl-lib:fref h-%data%
        (kwtop (f2cl-lib:int-sub kwtop 1)) ((1 ldh) (1 *))
        h-%offset%)))
  (cond

```

```

(= kbot kwtop)
(setf (f2cl-lib:fref sh-%data% (kwtop) ((1 *)) sh-%offset%)
      (f2cl-lib:fref h-%data% (kwtop kwtop) ((1 ldh) (1 *)) h-%offset%))
(setf ns 1) (setf nd 0)
(cond
 ((<= (cabs1 s)
      (max smlnum
           (* ulp (cabs1
                  (f2cl-lib:fref h (kwtop kwtop) ((1 ldh) (1 *)))))))
  (setf ns 0) (setf nd 1)
  (if (> kwtop ktop)
      (setf
       (f2cl-lib:fref h-%data% (kwtop (f2cl-lib:int-sub kwtop 1))
                       ((1 ldh) (1 *)) h-%offset%)
       zero))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
(go end_label)))

(zlacpy "U" jw jw
 (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 (kwtop kwtop)
                      ((1 ldh) (1 *)) h-%offset%)
 ldh t$ ldt)
(zcopy (f2cl-lib:int-sub jw 1)
 (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 ((+ kwtop 1) kwtop)
                      ((1 ldh) (1 *)) h-%offset%)
 (f2cl-lib:int-add ldh 1)
 (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (2 1)
                      ((1 ldt) (1 *))
                      t$-%offset%)
 (f2cl-lib:int-add ldt 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
 (zlaset "A" jw jw zero one v ldv) (declare (ignore var-0 var-5))
 (when var-1 (setf jw var-1)) (when var-2 (setf jw var-2))
 (when var-3 (setf zero var-3)) (when var-4 (setf one var-4))
 (when var-6 (setf ldv var-6)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10 var-11
  var-12)
 (zlahqr f2cl-lib:%true% f2cl-lib:%true% jw 1 jw t$ ldt
 (f2cl-lib:array-slice sh-%data% f2cl-lib:complex16 (kwtop) ((1 *))
                       sh-%offset%)
 1 jw v ldv infqr)
 (declare
  (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7 var-8 var-9 var-10
   var-11))
 (setf ldt var-6) (setf infqr var-12))
(setf ns jw) (setf ilst (f2cl-lib:int-add infqr 1))
(f2cl-lib:fdo (knt (f2cl-lib:int-add infqr 1)
                 (f2cl-lib:int-add knt 1))

```

```

      knt jw)
      nil)
      (tagbody
        (setf foo
          (cabs1 (f2cl-lib:fref t$-%data% (ns ns)
            ((1 ldt) (1 *)) t$-%offset%)))
          (if (= foo rzero) (setf foo (cabs1 s)))
          (cond
            ((<= (* (cabs1 s)
              (cabs1 (f2cl-lib:fref v (1 ns) ((1 ldv) (1 *))))
              (max smlnum (* ulp foo))))
              (setf ns (f2cl-lib:int-sub ns 1)))
            (t (setf ifst ns)
              (multiple-value-bind
                (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
                (ztrexc "V" jw t$ ldt v ldv ifst ilst info)
                (declare (ignore var-0 var-2 var-4))
                (when var-1 (setf jw var-1))
                (when var-3 (setf ldt var-3))
                (when var-5 (setf ldv var-5))
                (when var-6 (setf ifst var-6))
                (when var-7 (setf ilst var-7))
                (when var-8 (setf info var-8))
                (setf ilst (f2cl-lib:int-add ilst 1))))
              label10))
        (if (= ns 0) (setf s zero))
        (cond
          ((< ns jw)
            (f2cl-lib:fdo (i (f2cl-lib:int-add infqr 1) (f2cl-lib:int-add i 1))
              (> i
                ns)
              nil)
            (tagbody (setf ifst i)
              (f2cl-lib:fdo (j (f2cl-lib:int-add i 1)
                (f2cl-lib:int-add j 1))
                (> j ns)
                nil)
              (tagbody
                (if
                  (> (cabs1 (f2cl-lib:fref t$-%data% (j j)
                    ((1 ldt) (1 *)) t$-%offset%))
                    (cabs1
                      (f2cl-lib:fref t$-%data% (ifst ifst)
                        ((1 ldt) (1 *)) t$-%offset%)))
                    (setf ifst j))
                  label20))
                (setf ilst i)
                (if (/= ifst ilst)
                  (multiple-value-bind
                    (var-0 var-1 var-2 var-3 var-4 var-5 var-6

```

```

        var-7 var-8)
        (ztrexc "V" jw t$ ldt v ldv ifst ilst info)
        (declare (ignore var-0 var-2 var-4))
        (when var-1 (setf jw var-1))
        (when var-3 (setf ldt var-3))
        (when var-5 (setf ldv var-5))
        (when var-6 (setf ifst var-6))
        (when var-7 (setf ilst var-7))
        (when var-8 (setf info var-8))))
    label30))))
(f2cl-lib:fdo (i (f2cl-lib:int-add infqr 1) (f2cl-lib:int-add i 1))
  (> i jw)
  nil)
  (tagbody
    (setf
      (f2cl-lib:fref sh-%data%
        ((f2cl-lib:int-sub (f2cl-lib:int-add kwtop i) 1))
        ((1 *)) sh-%offset%)
      (f2cl-lib:fref t$-%data% (i i) ((1 ldt) (1 *))
        t$-%offset%))
    label40))
(cond
  ((or (< ns jw) (= s zero))
    (cond
      ((and (> ns 1) (/= s zero))
        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
          (zcopy ns v ldv work 1) (declare (ignore var-1 var-3 var-4))
          (when var-0 (setf ns var-0)) (when var-2 (setf ldv var-2)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i ns) nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data% (i) ((1 *))
              work-%offset%)
              (coerce
                (f2cl-lib:dconjg
                  (f2cl-lib:fref work-%data% (i) ((1 *))
                    work-%offset%))
                'f2cl-lib:complex16))
              label50))
            (setf beta (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (zlarfg ns beta
                (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (2) ((1 *))
                  work-%offset%)
                  1 tau)
                (declare (ignore var-2 var-3)) (when var-0 (setf ns var-0))
                (when var-1 (setf beta var-1)) (when var-4 (setf tau var-4)))
            (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
              (zlaset "L"

```

```

(f2cl-lib:int-sub jw 2) (f2cl-lib:int-sub jw 2) zero zero
(f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (3 1)
  ((1 ldt) (1 *)) t$-%offset%)
ldt)
(declare (ignore var-0 var-1 var-2 var-5))
(when var-3 (setf zero var-3)) (when var-4 (setf zero var-4))
(when var-6 (setf ldt var-6)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "L" ns jw work 1 (f2cl-lib:dconjg tau) t$ ldt
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
      ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-5 var-6 var-8))
  (when var-1 (setf ns var-1)) (when var-2 (setf jw var-2))
  (when var-7 (setf ldt var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "R" ns ns work 1 tau t$ ldt
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
      ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-6 var-8))
  (when var-1 (setf ns var-1)) (when var-2 (setf ns var-2))
  (when var-5 (setf tau var-5)) (when var-7 (setf ldt var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "R" jw ns work 1 tau v ldv
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
      ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-6 var-8))
  (when var-1 (setf jw var-1)) (when var-2 (setf ns var-2))
  (when var-5 (setf tau var-5)) (when var-7 (setf ldv var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zgehrd jw 1 ns t$ ldt work
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
      ((+ jw 1)) ((1 *))
      work-%offset%)
    (f2cl-lib:int-sub lwork jw) info)
  (declare (ignore var-1 var-3 var-5 var-6 var-7)) (setf jw var-0)
  (setf ns var-2) (setf ldt var-4) (setf info var-8)))
(if (> kwtop 1)
  (setf
    (f2cl-lib:fref h-%data% (kwtop (f2cl-lib:int-sub kwtop 1))
      ((1 ldh) (1 *)) h-%offset%)
  (coerce
    (* s
      (f2cl-lib:dconjg

```

```

      (f2cl-lib:fref v-%data% (1 1) ((1 ldv) (1 *)) v-%offset%)))
      'f2cl-lib:complex16)))
(zlacpy "U" jw jw t$ ldt
 (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 (kwtop kwtop)
  ((1 ldh) (1 *)) h-%offset%)
 ldh)
(zcopy (f2cl-lib:int-sub jw 1)
 (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
  (2 1) ((1 ldt) (1 *))
  t$-%offset%)
 (f2cl-lib:int-add ldt 1)
 (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
  ((+ kwtop 1) kwtop)
  ((1 ldh) (1 *)) h-%offset%)
 (f2cl-lib:int-add ldh 1))
(if (and (> ns 1) (/= s zero))
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10
   var-11 var-12 var-13)
  (zunmhr "R" "N" jw ns 1 ns t$ ldt work v ldv
   (f2cl-lib:array-slice work-%data% f2cl-lib:complex16
    ((+ jw 1) ((1 *)))
    work-%offset%)
   (f2cl-lib:int-sub lwork jw) info)
  (declare (ignore var-0 var-1 var-4 var-6 var-8 var-9 var-11
   var-12))
  (when var-2 (setf jw var-2)) (when var-3 (setf ns var-3))
  (when var-5 (setf ns var-5)) (when var-7 (setf ldt var-7))
  (when var-10 (setf ldv var-10)) (when var-13 (setf info var-13))))
(cond (wantt (setf ltop 1)) (t (setf ltop ktop)))
(f2cl-lib:fdo (krow ltop (f2cl-lib:int-add krow nv))
  ((> krow
   (f2cl-lib:int-add kwtop (f2cl-lib:int-sub 1)))
   nil)
 (tagbody
  (setf kln
   (min (the f2cl-lib:integer4 nv)
    (the f2cl-lib:integer4
     (f2cl-lib:int-sub kwtop krow)))))
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10
    var-11 var-12)
   (zgemm "N" "N" kln jw jw one
    (f2cl-lib:array-slice h-%data%
     f2cl-lib:complex16 (krow kwtop)
     ((1 ldh) (1 *)) h-%offset%)
    ldh v ldv zero wv ldwv)
   (declare (ignore var-0 var-1 var-6 var-8 var-11)))

```

```

        (when var-2 (setf kln var-2))
        (when var-3 (setf jw var-3))
        (when var-4 (setf jw var-4))
        (when var-5 (setf one var-5))
        (when var-7 (setf ldh var-7))
        (when var-9 (setf ldv var-9))
        (when var-10 (setf zero var-10))
        (when var-12 (setf ldwv var-12)))
(zlacpy "A" kln jw wv ldwv
 (f2cl-lib:array-slice h-%data%
  f2cl-lib:complex16 (krow kwtop)
  ((1 ldh) (1 *)) h-%offset%)
ldh)
label60))
(cond
 (wantt
  (f2cl-lib:fdo (kcol (f2cl-lib:int-add kbot 1)
    (f2cl-lib:int-add kcol nh))
    ((> kcol n) nil)
    (tagbody
      (setf kln
        (min (the f2cl-lib:integer4 nh)
          (the f2cl-lib:integer4
            (f2cl-lib:int-add
              (f2cl-lib:int-sub n kcol) 1))))))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6
         var-7 var-8 var-9 var-10
         var-11 var-12)
        (zgemm "C" "N" jw kln jw one v ldv
          (f2cl-lib:array-slice h-%data%
            f2cl-lib:complex16 (kwtop kcol)
            ((1 ldh) (1 *)) h-%offset%)
          ldh zero t$ ldt)
        (declare (ignore var-0 var-1 var-6 var-8 var-11))
        (when var-2 (setf jw var-2))
        (when var-3 (setf kln var-3))
        (when var-4 (setf jw var-4))
        (when var-5 (setf one var-5))
        (when var-7 (setf ldv var-7))
        (when var-9 (setf ldh var-9))
        (when var-10 (setf zero var-10))
        (when var-12 (setf ldt var-12)))
        (zlacpy "A" jw kln t$ ldt
          (f2cl-lib:array-slice h-%data%
            f2cl-lib:complex16 (kwtop kcol)
            ((1 ldh) (1 *)) h-%offset%)
          ldh)
        label70))))))
(cond

```



```

(wantz
  (f2cl-lib:fdo (krow iloz (f2cl-lib:int-add krow nv))
    (> krow ihiz)
    nil)
  (tagbody
    (setf kln
      (min (the f2cl-lib:integer4 nv)
        (the f2cl-lib:integer4
          (f2cl-lib:int-add
            (f2cl-lib:int-sub ihiz krow) 1))))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10
       var-11 var-12)
      (zgemm "N" "N" kln jw jw one
        (f2cl-lib:array-slice z-%data%
          f2cl-lib:complex16 (krow kwtop)
            ((1 ldz) (1 *)) z-%offset%)
        ldz v ldv zero wv ldwv)
      (declare (ignore var-0 var-1 var-6 var-8 var-11))
      (when var-2 (setf kln var-2))
      (when var-3 (setf jw var-3))
      (when var-4 (setf jw var-4))
      (when var-5 (setf one var-5))
      (when var-7 (setf ldz var-7))
      (when var-9 (setf ldv var-9))
      (when var-10 (setf zero var-10))
      (when var-12 (setf ldwv var-12)))
    (zlacpy "A" kln jw wv ldwv
      (f2cl-lib:array-slice z-%data%
        f2cl-lib:complex16 (krow kwtop)
          ((1 ldz) (1 *)) z-%offset%)
      ldz)
    label80))))))
(setf nd (f2cl-lib:int-sub jw ns))
(setf ns (f2cl-lib:int-sub ns infqr))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (f2cl-lib:dcmplx lwkopt 0))
end_label
(return
  (values nil nil nil nil nil nil nil ldh nil nil nil ldz ns nd nil
    nil ldv nil nil ldt nil nil ldwv nil nil))))))

```

zlaqr3 LAPACK

— zlaqr3.input —

```

)set break resume
)sys rm -f zlaqr3.output
)spool zlaqr3.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlaqr3.help —

```

=====
zlaqr3 examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```

SUBROUTINE ZLAQR3( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
                  IHIZ, Z, LDZ, NS, ND, SH, V, LDV, NH, T, LDT,
                  NV, WV, LDWV, WORK, LWORK )

```

.. Scalar Arguments ..

```

INTEGER          IHIZ, ILOZ, KBOT, KTOP, LDH, LDT, LDV, LDWV,
$                LDZ, LWORK, N, ND, NH, NS, NV, NW
LOGICAL          WANTT, WANTZ

```

..

.. Array Arguments ..

```

COMPLEX*16       H( LDH, * ), SH( * ), T( LDT, * ), V( LDV, * ),
$                WORK( * ), WV( LDWV, * ), Z( LDZ, * )

```

..

Purpose:

=====

Aggressive early deflation:

ZLAQR3 accepts as input an upper Hessenberg matrix H and performs an unitary similarity transformation designed to detect and deflate fully converged eigenvalues from a trailing principal submatrix. On output H has been overwritten by a new Hessenberg matrix that is a perturbation of an unitary similarity transformation of H. It is to be hoped that the final version of H has many zero subdiagonal entries.

Arguments:

=====

[in] WANTT

WANTT is LOGICAL

If .TRUE., then the Hessenberg matrix H is fully updated so that the triangular Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then only enough of H is updated to preserve the eigenvalues.

[in] WANTZ

WANTZ is LOGICAL

If .TRUE., then the unitary matrix Z is updated so so that the unitary Schur factor may be computed (in cooperation with the calling subroutine). If .FALSE., then Z is not referenced.

[in] N

N is INTEGER

The order of the matrix H and (if WANTZ is .TRUE.) the order of the unitary matrix Z.

[in] KTOP

KTOP is INTEGER

It is assumed that either KTOP = 1 or H(KTOP,KTOP-1)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

[in] KBOT

KBOT is INTEGER

It is assumed without a check that either KBOT = N or H(KBOT+1,KBOT)=0. KBOT and KTOP together determine an isolated block along the diagonal of the Hessenberg matrix.

[in] NW

NW is INTEGER
Deflation window size. 1 .LE. NW .LE. (KBOT-KTOP+1).

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)
On input the initial N-by-N section of H stores the
Hessenberg matrix undergoing aggressive early deflation.
On output H has been transformed by a unitary
similarity transformation, perturbed, and the returned
to Hessenberg form that (it is to be hoped) has some
zero subdiagonal entries.

[in] LDH

LDH is integer
Leading dimension of H just as declared in the calling
subroutine. N .LE. LDH

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER
Specify the rows of Z to which transformations must be
applied if WANTZ is .TRUE.. 1 .LE. ILOZ .LE. IHIZ .LE. N.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,N)
If WANTZ is .TRUE., then on output, the unitary
similarity transformation mentioned above has been
accumulated into Z(ILOZ:IHIZ,ILO:IHI) from the right.
If WANTZ is .FALSE., then Z is unreferenced.

[in] LDZ

LDZ is integer
The leading dimension of Z just as declared in the
calling subroutine. 1 .LE. LDZ.

[out] NS

NS is integer
The number of unconverged (ie approximate) eigenvalues

returned in SR and SI that may be used as shifts by the calling subroutine.

[out] ND

ND is integer
The number of converged eigenvalues uncovered by this subroutine.

[out] SH

SH is COMPLEX*16 array, dimension KBOT
On output, approximate eigenvalues that may be used for shifts are stored in SH(KBOT-ND-NS+1) through SR(KBOT-ND). Converged eigenvalues are stored in SH(KBOT-ND+1) through SH(KBOT).

[out] V

V is COMPLEX*16 array, dimension (LDV,NW)
An NW-by-NW work array.

[in] LDV

LDV is integer scalar
The leading dimension of V just as declared in the calling subroutine. NW .LE. LDV

[in] NH

NH is integer scalar
The number of columns of T. NH.GE.NW.

[out] T

T is COMPLEX*16 array, dimension (LDT,NW)

[in] LDT

LDT is integer
The leading dimension of T just as declared in the calling subroutine. NW .LE. LDT

[in] NV

NV is integer
The number of rows of work array WV available for workspace. NV.GE.NW.

[out] WV

WV is COMPLEX*16 array, dimension (LDWV,NW)

[in] LDWV

LDWV is integer
The leading dimension of W just as declared in the
calling subroutine. NW .LE. LDV

[out] WORK

WORK is COMPLEX*16 array, dimension LWORK.
On exit, WORK(1) is set to an estimate of the optimal value
of LWORK for the given values of N, NW, KTOP and KBOT.

[in] LWORK

LWORK is integer
The dimension of the work array WORK. LWORK = 2*NW
suffices, but greater efficiency may result from larger
values of LWORK.

If LWORK = -1, then a workspace query is assumed; ZLAQR3
only estimates the optimal workspace size for the given
values of N, NW, KTOP and KBOT. The estimate is returned
in WORK(1). No error message related to LWORK is issued
by XERBLA. Neither H nor Z are accessed.

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Contributors:

=====

Karen Braman and Ralph Byers, Department of Mathematics,
University of Kansas, USA

```

* =====
*      SUBROUTINE ZLAQR3( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
$              IHIZ, Z, LDZ, NS, ND, SH, V, LDV, NH, T, LDT,
$              NV, WV, LDWV, WORK, LWORK )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*   November 2011
*
* .. Scalar Arguments ..
*      INTEGER          IHIZ, ILOZ, KBOT, KTOP, LDH, LDT, LDV, LDWV,
$      LDZ, LWORK, N, ND, NH, NS, NV, NW
*      LOGICAL          WANTT, WANTZ
*
* ..
* .. Array Arguments ..
*      COMPLEX*16       H( LDH, * ), SH( * ), T( LDT, * ), V( LDV, * ),
$      WORK( * ), WV( LDWV, * ), Z( LDZ, * )
*
* ..
*
* =====
*
* .. Parameters ..
*      COMPLEX*16       ZERO, ONE
*      PARAMETER        ( ZERO = ( 0.0d0, 0.0d0 ),
$      ONE = ( 1.0d0, 0.0d0 ) )
*      DOUBLE PRECISION RZERO, RONE
*      PARAMETER        ( RZERO = 0.0d0, RONE = 1.0d0 )
*
* ..
* .. Local Scalars ..
*      COMPLEX*16       BETA, CDUM, S, TAU
*      DOUBLE PRECISION FOO, SAFMAX, SAFMIN, SMLNUM, ULP
*      INTEGER          I, IFST, ILST, INFO, INFQR, J, JW, KCOL, KLN,
$      KNT, KROW, KWTOP, LTOP, LWK1, LWK2, LWK3,
$      LWKOPT, NMIN
*
* ..
* .. External Functions ..
*      DOUBLE PRECISION DLAMCH
*      INTEGER          ILAENV
*      EXTERNAL         DLAMCH, ILAENV
*
* ..
* .. External Subroutines ..
*      EXTERNAL         DLABAD, ZCOPY, ZGHERD, ZGEMM, ZLACPY, ZLAHQR,
$      ZLAQR4, ZLARF, ZLARFG, ZLASET, ZTREXC, ZUMHR
*
* ..
* .. Intrinsic Functions ..
*      INTRINSIC        ABS, DBLE, DCMPLX, DCONJG, DIMAG, INT, MAX, MIN
*
* ..
* .. Statement Functions ..
*      DOUBLE PRECISION CABS1

```

```

*      ..
*      .. Statement Function definitions ..
*      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*      ..
*      .. Executable Statements ..
*
*      ==== Estimate optimal workspace. ====
*
*      JW = MIN( NW, KBOT-KTOP+1 )
*      IF( JW.LE.2 ) THEN
*          LWKOPT = 1
*      ELSE
*
*          ==== Workspace query call to ZGEHRD ====
*
*          CALL ZGEHRD( JW, 1, JW-1, T, LDT, WORK, WORK, -1, INFO )
*          LWK1 = INT( WORK( 1 ) )
*
*          ==== Workspace query call to ZUNMHR ====
*
*          CALL ZUNMHR( 'R', 'N', JW, JW, 1, JW-1, T, LDT, WORK, V, LDV,
$              WORK, -1, INFO )
*          LWK2 = INT( WORK( 1 ) )
*
*          ==== Workspace query call to ZLAQR4 ====
*
*          CALL ZLAQR4( .true., .true., JW, 1, JW, T, LDT, SH, 1, JW, V,
$              LDV, WORK, -1, INFQR )
*          LWK3 = INT( WORK( 1 ) )
*
*          ==== Optimal workspace ====
*
*          LWKOPT = MAX( JW+MAX( LWK1, LWK2 ), LWK3 )
*      END IF
*
*      ==== Quick return in case of workspace query. ====
*
*      IF( LWORK.EQ.-1 ) THEN
*          WORK( 1 ) = DCMPLX( LWKOPT, 0 )
*          RETURN
*      END IF
*
*      ==== Nothing to do ...
*      ... for an empty active block ... ====
*      NS = 0
*      ND = 0
*      WORK( 1 ) = ONE
*      IF( KTOP.GT.KBOT )
$      RETURN
*      ... nor for an empty deflation window. ====

```



```

      IF( NW.LT.1 )
$      RETURN
*
*      ==== Machine constants ====
*
      SAFMIN = DLAMCH( 'SAFE MINIMUM' )
      SAFMAX = RONE / SAFMIN
      CALL DLABAD( SAFMIN, SAFMAX )
      ULP = DLAMCH( 'PRECISION' )
      SMLNUM = SAFMIN*( DBLE( N ) / ULP )
*
*      ==== Setup deflation window ====
*
      JW = MIN( NW, KBOT-KTOP+1 )
      KWTOP = KBOT - JW + 1
      IF( KWTOP.EQ.KTOP ) THEN
          S = ZERO
      ELSE
          S = H( KWTOP, KWTOP-1 )
      END IF
*
      IF( KBOT.EQ.KWTOP ) THEN
*
*          ==== 1-by-1 deflation window: not much to do ====
*
          SH( KWTOP ) = H( KWTOP, KWTOP )
          NS = 1
          ND = 0
          IF( CABS1( S ).LE.MAX( SMLNUM, ULP*CABS1( H( KWTOP,
$              KWTOP ) ) ) ) THEN
              NS = 0
              ND = 1
              IF( KWTOP.GT.KTOP )
$                  H( KWTOP, KWTOP-1 ) = ZERO
          END IF
          WORK( 1 ) = ONE
          RETURN
      END IF
*
*      ==== Convert to spike-triangular form. (In case of a
*      .   rare QR failure, this routine continues to do
*      .   aggressive early deflation using that part of
*      .   the deflation window that converged using INFQR
*      .   here and there to keep track.) ====
*
      CALL ZLACPY( 'U', JW, JW, H( KWTOP, KWTOP ), LDH, T, LDT )
      CALL ZCOPY( JW-1, H( KWTOP+1, KWTOP ), LDH+1, T( 2, 1 ), LDT+1 )
*
      CALL ZLASET( 'A', JW, JW, ZERO, ONE, V, LDV )
      NMIN = ILAENV( 12, 'ZLAQR3', 'SV', JW, 1, JW, LWORK )

```

```

      IF( JW.GT.NMIN ) THEN
        CALL ZLAQR4( .true., .true., JW, 1, JW, T, LDT, SH( KWTOP ), 1,
$              JW, V, LDV, WORK, LWORK, INFQR )
      ELSE
        CALL ZLAHQR( .true., .true., JW, 1, JW, T, LDT, SH( KWTOP ), 1,
$              JW, V, LDV, INFQR )
      END IF
*
*      ==== Deflation detection loop ====
*
      NS = JW
      ILST = INFQR + 1
      DO 10 KNT = INFQR + 1, JW
*
*      ==== Small spike tip deflation test ====
*
        FOO = CABS1( T( NS, NS ) )
        IF( FOO.EQ.RZERO )
$          FOO = CABS1( S )
        IF( CABS1( S )*CABS1( V( 1, NS ) ) .LE. MAX( SMLNUM, ULP*FOO ) )
$          THEN
*
*          ==== One more converged eigenvalue ====
*
          NS = NS - 1
        ELSE
*
*          ==== One undeflatable eigenvalue. Move it up out of the
*          . way. (ZTREXC can not fail in this case.) ====
*
          IFST = NS
          CALL ZTREXC( 'V', JW, T, LDT, V, LDV, IFST, ILST, INFO )
          ILST = ILST + 1
        END IF
      10 CONTINUE
*
*      ==== Return to Hessenberg form ====
*
      IF( NS.EQ.0 )
$        S = ZERO
*
      IF( NS.LT.JW ) THEN
*
*      ==== sorting the diagonal of T improves accuracy for
*      . graded matrices. ====
*
      DO 30 I = INFQR + 1, NS
        IFST = I
        DO 20 J = I + 1, NS
          IF( CABS1( T( J, J ) ) .GT. CABS1( T( IFST, IFST ) ) )

```

```

$          IFST = J
20      CONTINUE
        ILST = I
        IF( IFST.NE.ILST )
$          CALL ZTREXC( 'V', JW, T, LDT, V, LDV, IFST, ILST, INFO )
30      CONTINUE
        END IF
*
*      ==== Restore shift/eigenvalue array from T ====
*
        DO 40 I = INFQR + 1, JW
            SH( KWTOP+I-1 ) = T( I, I )
40      CONTINUE
*
*
        IF( NS.LT.JW .OR. S.EQ.ZERO ) THEN
            IF( NS.GT.1 .AND. S.NE.ZERO ) THEN
*
*          ==== Reflect spike back into lower triangle ====
*
                CALL ZCOPY( NS, V, LDV, WORK, 1 )
                DO 50 I = 1, NS
                    WORK( I ) = DCONJG( WORK( I ) )
50          CONTINUE
                BETA = WORK( 1 )
                CALL ZLARFG( NS, BETA, WORK( 2 ), 1, TAU )
                WORK( 1 ) = ONE
*
*          CALL ZLASET( 'L', JW-2, JW-2, ZERO, ZERO, T( 3, 1 ), LDT )
*
                CALL ZLARF( 'L', NS, JW, WORK, 1, DCONJG( TAU ), T, LDT,
$                  WORK( JW+1 ) )
                CALL ZLARF( 'R', NS, NS, WORK, 1, TAU, T, LDT,
$                  WORK( JW+1 ) )
                CALL ZLARF( 'R', JW, NS, WORK, 1, TAU, V, LDV,
$                  WORK( JW+1 ) )
*
                CALL ZGHRD( JW, 1, NS, T, LDT, WORK, WORK( JW+1 ),
$                  LWORK-JW, INFO )
            END IF
*
*      ==== Copy updated reduced window into place ====
*
            IF( KWTOP.GT.1 )
$              H( KWTOP, KWTOP-1 ) = S*DCONJG( V( 1, 1 ) )
                CALL ZLACPY( 'U', JW, JW, T, LDT, H( KWTOP, KWTOP ), LDH )
                CALL ZCOPY( JW-1, T( 2, 1 ), LDT+1, H( KWTOP+1, KWTOP ),
$                  LDH+1 )
*
*      ==== Accumulate orthogonal matrix in order update

```

```

*      .      H and Z, if requested.  ====
*
      IF( NS.GT.1 .AND. S.NE.ZERO )
$      CALL ZUNMHR( 'R', 'N', JW, NS, 1, NS, T, LDT, WORK, V, LDV,
$                  WORK( JW+1 ), LWORK-JW, INFO )
*
*      ==== Update vertical slab in H ====
*
      IF( WANTT ) THEN
        LTOP = 1
      ELSE
        LTOP = KTOP
      END IF
      DO 60 KROW = LTOP, KWTOP - 1, NV
        KLN = MIN( NV, KWTOP-KROW )
        CALL ZGEMM( 'N', 'N', KLN, JW, JW, ONE, H( KROW, KWTOP ),
$                LDH, V, LDV, ZERO, WV, LDWV )
        CALL ZLACPY( 'A', KLN, JW, WV, LDWV, H( KROW, KWTOP ), LDH )
60      CONTINUE
*
*      ==== Update horizontal slab in H ====
*
      IF( WANTT ) THEN
        DO 70 KCOL = KBOT + 1, N, NH
          KLN = MIN( NH, N-KCOL+1 )
          CALL ZGEMM( 'C', 'N', JW, KLN, JW, ONE, V, LDV,
$                H( KWTOP, KCOL ), LDH, ZERO, T, LDT )
          CALL ZLACPY( 'A', JW, KLN, T, LDT, H( KWTOP, KCOL ),
$                LDH )
70      CONTINUE
      END IF
*
*      ==== Update vertical slab in Z ====
*
      IF( WANTZ ) THEN
        DO 80 KROW = ILOZ, IHIZ, NV
          KLN = MIN( NV, IHIZ-KROW+1 )
          CALL ZGEMM( 'N', 'N', KLN, JW, JW, ONE, Z( KROW, KWTOP ),
$                LDZ, V, LDV, ZERO, WV, LDWV )
          CALL ZLACPY( 'A', KLN, JW, WV, LDWV, Z( KROW, KWTOP ),
$                LDZ )
80      CONTINUE
      END IF
      END IF
*
*      ==== Return the number of deflations ... ====
*
      ND = JW - NS
*
*      ==== ... and the number of shifts. (Subtracting

```

```

*      .   INFQR from the spike length takes care
*      .   of the case of a rare QR failure while
*      .   calculating eigenvalues of the deflation
*      .   window.) ====
*
      NS = NS - INFQR
*
*      ==== Return optimal workspace. ====
*
      WORK( 1 ) = DCMPLX( LWKOPT, 0 )
*
*      ==== End of ZLAQR3 ====
*
      END

```

— LAPACK zlaqr3 —

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (rzero 0.0d0)
   (rone 1.0d0))
  (declare (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
            (type (double-float 0.0d0 0.0d0) rzero)
            (type (double-float 1.0d0 1.0d0) rone) (ignorable zero one rzero rone))
  (defun zlaqr3
    (wantt wantz n ktop kbot nw h ldh iloz ihiz z ldz ns nd sh v ldv nh t$ ldt nv
     wv ldwv work lwork)
    (declare (type f2cl-lib:logical wantt wantz)
              (type (f2cl-lib:integer4) lwork ldwv nv ldt nh ldv nd ns ldz ihiz iloz ldh
                    nw kbot ktop n)
              (type (array f2cl-lib:complex16 (*)) work wv t$ v sh z h))
    (f2cl-lib:with-multi-array-data
      ((h f2cl-lib:complex16 h-%data% h-%offset%)
       (z f2cl-lib:complex16 z-%data% z-%offset%)
       (sh f2cl-lib:complex16 sh-%data% sh-%offset%)
       (v f2cl-lib:complex16 v-%data% v-%offset%)
       (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
       (wv f2cl-lib:complex16 wv-%data% wv-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (labels
        ((cabs1 (cdum)
          (+ (abs (f2cl-lib:dbler cdum)) (abs (f2cl-lib:dimag cdum)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                           (values double-float &rest t)) cabs1))
        (prog

```

```

((i 0) (ifst 0) (ilst 0) (info 0) (infqr 0) (j 0) (jw 0)
 (kcol 0) (kln 0)
 (knt 0) (krow 0) (kwtop 0) (ltop 0) (lwk1 0) (lwk2 0)
 (lwk3 0) (lwkoft 0)
 (nmin 0) (foo 0.0d0) (safmax 0.0d0) (safmin 0.0d0) (smlnum 0.0d0)
 (ulp 0.0d0) (beta #C(0.0d0 0.0d0)) (cdum #C(0.0d0 0.0d0))
 (s #C(0.0d0 0.0d0)) (tau #C(0.0d0 0.0d0)) (dconjg$ 0.0))
(declare
 (type (f2cl-lib:integer4) nmin lwkoft lwk3 lwk2 lwk1 ltop
  kwtop krow knt kln
  kcol jw j infqr info ilst ifst i)
 (type (double-float) ulp smlnum safmin safmax foo)
 (type (f2cl-lib:complex16) tau s cdum beta)
 (type (single-float) dconjg$))
(setf jw
 (min (the f2cl-lib:integer4 nw)
  (the f2cl-lib:integer4
   (f2cl-lib:int-add (f2cl-lib:int-sub kbot ktop) 1))))
(cond ((<= jw 2) (setf lwkoft 1))
 (t
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
   (zgehrd jw 1 (f2cl-lib:int-sub jw 1) t$ ldt work work -1 info)
   (declare (ignore var-1 var-2 var-3 var-5 var-6 var-7))
   (setf jw var-0)
   (setf ldt var-4) (setf info var-8))
  (setf lwk1
   (f2cl-lib:int
    (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)))
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10 var-11
    var-12 var-13)
   (zunmhr "R" "N" jw jw 1
    (f2cl-lib:int-sub jw 1) t$ ldt work v ldv work -1
    info)
   (declare (ignore var-0 var-1 var-4 var-5 var-6 var-8
    var-9 var-11 var-12))
   (when var-2 (setf jw var-2)) (when var-3 (setf jw var-3))
   (when var-7 (setf ldt var-7)) (when var-10 (setf ldv var-10))
   (when var-13 (setf info var-13)))
  (setf lwk2
   (f2cl-lib:int
    (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)))
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
    var-9 var-10 var-11
    var-12 var-13 var-14)
   (zlaqr4 f2cl-lib:%true% f2cl-lib:%true%
    jw 1 jw t$ ldt sh 1 jw v ldv work

```

```

-1 infqr)
(declare
  (ignore var-0 var-1 var-3 var-5 var-7 var-8 var-10 var-12 var-13))
(when var-2 (setf jw var-2)) (when var-4 (setf jw var-4))
(when var-6 (setf ldt var-6)) (when var-9 (setf jw var-9))
(when var-11 (setf ldv var-11)) (when var-14 (setf infqr var-14)))
(setf lwk3
  (f2cl-lib:int (f2cl-lib:fref work-%data% (1) ((1 *))
    work-%offset%)))
(setf lwkopt
  (max
    (the f2cl-lib:integer4
      (f2cl-lib:int-add jw
        (max (the f2cl-lib:integer4 lwk1)
          (the f2cl-lib:integer4 lwk2))))
    (the f2cl-lib:integer4 lwk3))))
(cond
  ((= lwork (f2cl-lib:int-sub 1))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (f2cl-lib:dcmplx lwkopt 0))
    (go end_label)))
(setf ns 0) (setf nd 0)
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
(if (> ktop kbot) (go end_label)) (if (< nw 1) (go end_label))
(setf safmin (dlamch "SAFE MINIMUM")) (setf safmax (/ rone safmin))
(multiple-value-bind (var-0 var-1)
  (dlabad safmin safmax) (declare (ignore))
  (when var-0 (setf safmin var-0)) (when var-1 (setf safmax var-1)))
(setf ulp (dlamch "PRECISION"))
(setf smlnum (* safmin (/ (f2cl-lib:dbple n) ulp)))
(setf jw
  (min (the f2cl-lib:integer4 nw)
    (the f2cl-lib:integer4
      (f2cl-lib:int-add (f2cl-lib:int-sub kbot ktop) 1))))
(setf kwtop (f2cl-lib:int-add (f2cl-lib:int-sub kbot jw) 1))
(cond ((= kwtop ktop) (setf s zero))
  (t
    (setf s
      (f2cl-lib:fref h-%data%
        (kwtop (f2cl-lib:int-sub kwtop 1)) ((1 ldh) (1 *))
        h-%offset%)))
  (cond
    ((= kbot kwtop)
      (setf (f2cl-lib:fref sh-%data% (kwtop) ((1 *)) sh-%offset%)
        (f2cl-lib:fref h-%data% (kwtop kwtop) ((1 ldh) (1 *)) h-%offset%))
      (setf ns 1) (setf nd 0)
      (cond
        ((<= (cabs1 s)
          (max smlnum
            (* ulp (cabs1

```

```

        (f2cl-lib:fref h (kwtop kwtop) ((1 ldh) (1 *))))))
(setf ns 0) (setf nd 1)
(if (> kwtop ktop)
    (setf
      (f2cl-lib:fref h-%data% (kwtop (f2cl-lib:int-sub kwtop 1))
        ((1 ldh) (1 *)) h-%offset%)
      zero)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
(go end_label)))
(zlscopy "U" jw jw
  (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 (kwtop kwtop)
    ((1 ldh) (1 *)) h-%offset%)
  ldh t$ ldt)
(zcopy (f2cl-lib:int-sub jw 1)
  (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 ((+ kwtop 1) kwtop)
    ((1 ldh) (1 *)) h-%offset%)
  (f2cl-lib:int-add ldh 1)
  (f2cl-lib:array-slice t$-%data%
    f2cl-lib:complex16 (2 1) ((1 ldt) (1 *))
    t$-%offset%)
  (f2cl-lib:int-add ldt 1))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (zlaset "A" jw jw zero one v ldv) (declare (ignore var-0 var-5))
  (when var-1 (setf jw var-1)) (when var-2 (setf jw var-2))
  (when var-3 (setf zero var-3)) (when var-4 (setf one var-4))
  (when var-6 (setf ldv var-6)))
(setf nmin
  (multiple-value-bind (ret-val var-0 var-1 var-2 var-3
    var-4 var-5 var-6)
    (ilaenv 12 "ZLAQR3" "SV" jw 1 jw lwork)
    (declare (ignore var-0 var-1 var-2 var-4))
    (when var-3 (setf jw var-3))
    (when var-5 (setf jw var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(cond
  ((> jw nmin)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10 var-11
        var-12 var-13 var-14)
      (zlaqr4 f2cl-lib:%true% f2cl-lib:%true% jw 1 jw t$ ldt
        (f2cl-lib:array-slice sh-%data% f2cl-lib:complex16 (kwtop) ((1 *))
          sh-%offset%)
        1 jw v ldv work lwork infqr)
      (declare (ignore var-0 var-1 var-3 var-5 var-7 var-8
        var-10 var-12))
      (when var-2 (setf jw var-2)) (when var-4 (setf jw var-4))
      (when var-6 (setf ldt var-6)) (when var-9 (setf jw var-9))
      (when var-11 (setf ldv var-11)) (when var-13 (setf lwork var-13))
      (when var-14 (setf infqr var-14))))))

```



```

(t
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
     var-9 var-10 var-11
     var-12)
    (zlahqr f2cl-lib:%true% f2cl-lib:%true% jw 1 jw t$ ldt
      (f2cl-lib:array-slice sh-%data% f2cl-lib:complex16 (kwttop) ((1 *))
        sh-%offset%)
      1 jw v ldv infqr)
    (declare
      (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7 var-8
        var-9 var-10
        var-11))
    (setf ldt var-6) (setf infqr var-12))))
(setf ns jw) (setf ilst (f2cl-lib:int-add infqr 1))
(f2cl-lib:fdo (knt (f2cl-lib:int-add infqr 1)
  (f2cl-lib:int-add knt 1))
  ((>
    knt jw)
    nil)
  (tagbody
    (setf foo
      (cabs1 (f2cl-lib:fref t$-%data% (ns ns)
        ((1 ldt) (1 *)) t$-%offset%)))
    (if (= foo rzero) (setf foo (cabs1 s)))
    (cond
      ((<= (* (cabs1 s)
        (cabs1 (f2cl-lib:fref v (1 ns) ((1 ldv) (1 *))))
        (max smlnum (* ulp foo))))
      (setf ns (f2cl-lib:int-sub ns 1)))
    (t (setf ifst ns)
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
        (ztrexc "V" jw t$ ldt v ldv ifst ilst info)
        (declare (ignore var-0 var-2 var-4))
        (when var-1 (setf jw var-1))
        (when var-3 (setf ldt var-3))
        (when var-5 (setf ldv var-5))
        (when var-6 (setf ifst var-6))
        (when var-7 (setf ilst var-7))
        (when var-8 (setf info var-8)))
        (setf ilst (f2cl-lib:int-add ilst 1))))
      label10))
  (if (= ns 0) (setf s zero))
  (cond
    ((< ns jw)
      (f2cl-lib:fdo (i (f2cl-lib:int-add infqr 1) (f2cl-lib:int-add i 1))
        ((> i
          ns)
          nil)

```

```

(tagbody (setf ifst i)
  (f2cl-lib:fdo
    (j (f2cl-lib:int-add i 1) (f2cl-lib:int-add j 1))
    ((> j ns)
      nil)
    (tagbody
      (if
        (> (cabs1 (f2cl-lib:fref t$-%data% (j j)
          ((1 ldt) (1 *)) t$-%offset%))
          (cabs1
            (f2cl-lib:fref t$-%data% (ifst ifst)
              ((1 ldt) (1 *)) t$-%offset%)))
          (setf ifst j))
        label20))
      (setf ilst i)
      (if (/= ifst ilst)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6
            var-7 var-8)
          (ztrexc "V" jw t$ ldt v ldv ifst ilst info)
          (declare (ignore var-0 var-2 var-4))
          (when var-1 (setf jw var-1))
          (when var-3 (setf ldt var-3))
          (when var-5 (setf ldv var-5))
          (when var-6 (setf ifst var-6))
          (when var-7 (setf ilst var-7))
          (when var-8 (setf info var-8))))
        label30))))
(f2cl-lib:fdo (i (f2cl-lib:int-add infqr 1) (f2cl-lib:int-add i 1))
  ((> i jw)
    nil)
  (tagbody
    (setf
      (f2cl-lib:fref sh-%data%
        ((f2cl-lib:int-sub (f2cl-lib:int-add kwttop i) 1))
        ((1 *)) sh-%offset%)
      (f2cl-lib:fref t$-%data% (i i) ((1 ldt) (1 *))
        t$-%offset%))
    label40))
(cond
  ((or (< ns jw) (= s zero))
    (cond
      ((and (> ns 1) (/= s zero))
        (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
          (zcopy ns v ldv work 1) (declare (ignore var-1 var-3 var-4))
          (when var-0 (setf ns var-0)) (when var-2 (setf ldv var-2)))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          ((> i ns) nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data% (i) ((1 *))

```

```

        work-%offset%)
      (coerce
        (f2cl-lib:dconjg
          (f2cl-lib:fref work-%data% (i) ((1 *))
            work-%offset%))
        'f2cl-lib:complex16))
      label50))
(setf beta (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%))
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (zlarfg ns beta
    (f2cl-lib:array-slice work-%data% f2cl-lib:complex16 (2) ((1 *))
      work-%offset%)
    1 tau)
  (declare (ignore var-2 var-3)) (when var-0 (setf ns var-0))
  (when var-1 (setf beta var-1)) (when var-4 (setf tau var-4)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
  (zlaset "L"
    (f2cl-lib:int-sub jw 2) (f2cl-lib:int-sub jw 2) zero zero
    (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (3 1)
      ((1 ldt) (1 *)) t$-%offset%)
    ldt)
  (declare (ignore var-0 var-1 var-2 var-5))
  (when var-3 (setf zero var-3)) (when var-4 (setf zero var-4))
  (when var-6 (setf ldt var-6)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "L" ns jw work 1 (f2cl-lib:dconjg tau) t$ ldt
    (f2cl-lib:array-slice work-%data%
      f2cl-lib:complex16 ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-5 var-6 var-8))
  (when var-1 (setf ns var-1)) (when var-2 (setf jw var-2))
  (when var-7 (setf ldt var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "R" ns ns work 1 tau t$ ldt
    (f2cl-lib:array-slice work-%data%
      f2cl-lib:complex16 ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-6 var-8))
  (when var-1 (setf ns var-1)) (when var-2 (setf ns var-2))
  (when var-5 (setf tau var-5)) (when var-7 (setf ldt var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zlarf "R" jw ns work 1 tau v ldv
    (f2cl-lib:array-slice work-%data%
      f2cl-lib:complex16 ((+ jw 1)) ((1 *))
      work-%offset%))
  (declare (ignore var-0 var-3 var-4 var-6 var-8))

```

```

    (when var-1 (setf jw var-1)) (when var-2 (setf ns var-2))
    (when var-5 (setf tau var-5)) (when var-7 (setf ldv var-7)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
  (zgehrd jw 1 ns t$ ldt work
    (f2cl-lib:array-slice work-%data%
      f2cl-lib:complex16 ((+ jw 1)) ((1 *))
      work-%offset%)
    (f2cl-lib:int-sub lwork jw) info)
  (declare (ignore var-1 var-3 var-5 var-6 var-7)) (setf jw var-0)
  (setf ns var-2) (setf ldt var-4) (setf info var-8))))
(if (> kwtop 1)
  (setf
    (f2cl-lib:fref h-%data% (kwtop (f2cl-lib:int-sub kwtop 1))
      ((1 ldh) (1 *)) h-%offset%)
    (coerce
      (* s
        (f2cl-lib:dconjg
          (f2cl-lib:fref v-%data% (1 1) ((1 ldv) (1 *)) v-%offset%)))
      'f2cl-lib:complex16)))
  (zlacpy "U" jw jw t$ ldt
    (f2cl-lib:array-slice h-%data% f2cl-lib:complex16 (kwtop kwtop)
      ((1 ldh) (1 *)) h-%offset%)
    ldh)
  (zcopy (f2cl-lib:int-sub jw 1)
    (f2cl-lib:array-slice t$-%data%
      f2cl-lib:complex16 (2 1) ((1 ldt) (1 *))
      t$-%offset%)
    (f2cl-lib:int-add ldt 1)
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 ((+ kwtop 1) kwtop)
      ((1 ldh) (1 *)) h-%offset%)
    (f2cl-lib:int-add ldh 1))
  (if (and (> ns 1) (/= s zero))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
        var-9 var-10
        var-11 var-12 var-13)
      (zunmhr "R" "N" jw ns 1 ns t$ ldt work v ldv
        (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 ((+ jw 1)) ((1 *))
          work-%offset%)
        (f2cl-lib:int-sub lwork jw) info)
      (declare
        (ignore var-0 var-1 var-4 var-6 var-8 var-9 var-11 var-12))
      (when var-2 (setf jw var-2)) (when var-3 (setf ns var-3))
      (when var-5 (setf ns var-5)) (when var-7 (setf ldt var-7))
      (when var-10 (setf ldv var-10)) (when var-13 (setf info var-13))))
    (cond (wantt (setf ltop 1)) (t (setf ltop ktop)))
    (f2cl-lib:fdo (krow ltop (f2cl-lib:int-add krow nv))

```

```

(> krow
  (f2cl-lib:int-add kwtot (f2cl-lib:int-sub 1)))
  nil)
(tagbody
  (setf kln
    (min (the f2cl-lib:integer4 nv)
      (the f2cl-lib:integer4
        (f2cl-lib:int-sub kwtot krow)))))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6
     var-7 var-8 var-9 var-10
     var-11 var-12)
    (zgemm "N" "N" kln jw jw one
      (f2cl-lib:array-slice h-%data%
        f2cl-lib:complex16 (krow kwtot)
        ((1 ldh) (1 *)) h-%offset%)
      ldh v ldv zero wv ldwv)
    (declare (ignore var-0 var-1 var-6 var-8 var-11))
    (when var-2 (setf kln var-2))
    (when var-3 (setf jw var-3))
    (when var-4 (setf jw var-4))
    (when var-5 (setf one var-5))
    (when var-7 (setf ldh var-7))
    (when var-9 (setf ldv var-9))
    (when var-10 (setf zero var-10))
    (when var-12 (setf ldwv var-12)))
  (zlacpy "A" kln jw wv ldwv
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (krow kwtot)
      ((1 ldh) (1 *)) h-%offset%)
    ldh)
  label60))
(cond
  (wantt
    (f2cl-lib:fdo (kcol (f2cl-lib:int-add kbot 1)
      (f2cl-lib:int-add kcol nh))
      ((> kcol n) nil)
    (tagbody
      (setf kln
        (min (the f2cl-lib:integer4 nh)
          (the f2cl-lib:integer4
            (f2cl-lib:int-add
              (f2cl-lib:int-sub n kcol) 1)))))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6
         var-7 var-8 var-9 var-10
         var-11 var-12)
        (zgemm "C" "N" jw kln jw one v ldv
          (f2cl-lib:array-slice h-%data%
            f2cl-lib:complex16 (kwtot kcol)

```

```

      ((1 ldh) (1 *)) h-%offset%)
      ldh zero t$ ldt)
      (declare (ignore var-0 var-1 var-6 var-8 var-11))
      (when var-2 (setf jw var-2))
      (when var-3 (setf kln var-3))
      (when var-4 (setf jw var-4))
      (when var-5 (setf one var-5))
      (when var-7 (setf ldv var-7))
      (when var-9 (setf ldh var-9))
      (when var-10 (setf zero var-10))
      (when var-12 (setf ldt var-12)))
      (zlacpy "A" jw kln t$ ldt
      (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (kwttop kcol)
      ((1 ldh) (1 *)) h-%offset%)
      ldh)
      label70))))
(cond
  (wantz
    (f2cl-lib:fdo (krow iloz (f2cl-lib:int-add krow nv))
      (> krow ihiz)
      nil)
    (tagbody
      (setf kln
        (min (the f2cl-lib:integer4 nv)
          (the f2cl-lib:integer4
            (f2cl-lib:int-add
              (f2cl-lib:int-sub ihiz krow) 1))))))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10
       var-11 var-12)
      (zgemm "N" "N" kln jw jw one
        (f2cl-lib:array-slice z-%data%
          f2cl-lib:complex16 (krow kwttop)
          ((1 ldz) (1 *)) z-%offset%)
        ldz v ldv zero wv ldwv)
      (declare (ignore var-0 var-1 var-6 var-8 var-11))
      (when var-2 (setf kln var-2))
      (when var-3 (setf jw var-3))
      (when var-4 (setf jw var-4))
      (when var-5 (setf one var-5))
      (when var-7 (setf ldz var-7))
      (when var-9 (setf ldv var-9))
      (when var-10 (setf zero var-10))
      (when var-12 (setf ldwv var-12)))
      (zlacpy "A" kln jw wv ldwv
      (f2cl-lib:array-slice z-%data%
      f2cl-lib:complex16 (krow kwttop)
      ((1 ldz) (1 *)) z-%offset%)

```

```

                                ldz)
                                label80))))))
(setf nd (f2cl-lib:int-sub jw ns))
(setf ns (f2cl-lib:int-sub ns infqr))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (f2cl-lib:dcmplx lwkopt 0))
end_label
(return
 (values nil nil nil nil nil nil nil ldh nil nil nil ldz ns nd nil nil ldv
         nil nil ldt nil nil ldwv nil lwork))))))

```

zlaqr4 LAPACK

— zlaqr4.input —

```

)set break resume
)sys rm -f zlaqr4.output
)spool zlaqr4.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlaqr4.help —

```

=====
zlaqr4 examples
=====

=====
Man Page Details
=====

Online html documentation available at
      http://www.netlib.org/lapack/explore-html/

Definition:
=====

```

```

SUBROUTINE ZLAQR4( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,

```

```

                                IHIZ, Z, LDZ, WORK, LWORK, INFO )

.. Scalar Arguments ..
INTEGER                IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, LWORK, N
LOGICAL                WANTT, WANTZ
..
.. Array Arguments ..
COMPLEX*16            H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
..

```

Purpose:

=====

ZLAQR4 implements one level of recursion for ZLAQR0. It is a complete implementation of the small bulge multi-shift QR algorithm. It may be called by ZLAQR0 and, for large enough deflation window size, it may be called by ZLAQR3. This subroutine is identical to ZLAQR0 except that it calls ZLAQR2 instead of ZLAQR3.

ZLAQR4 computes the eigenvalues of a Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z T Z^*H$, where T is an upper triangular matrix (the Schur form), and Z is the unitary matrix of Schur vectors.

Optionally Z may be postmultiplied into an input unitary matrix Q so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the unitary matrix Q : $A = Q^*H^*Q^{**}H = (QZ)^*H^*(QZ)^{**}H$.

Arguments:

=====

[in] WANTT

WANTT is LOGICAL
 = .TRUE. : the full Schur form T is required;
 = .FALSE.: only eigenvalues are required.

[in] WANTZ

WANTZ is LOGICAL
 = .TRUE. : the matrix of Schur vectors Z is required;
 = .FALSE.: Schur vectors are not required.

[in] N

N is INTEGER
 The order of the matrix H . $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

It is assumed that H is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N and, if ILO.GT.1, H(ILO,ILO-1) is zero. ILO and IHI are normally set by a previous call to ZGEBAL, and then passed to ZGEHRD when the matrix output by ZGEBAL is reduced to Hessenberg form. Otherwise, ILO and IHI should be set to 1 and N, respectively. If N.GT.0, then 1.LE.ILO.LE.IHI.LE.N. If N = 0, then ILO = 1 and IHI = 0.

[in,out] H

H is COMPLEX*16 array, dimension (LDH,N)

On entry, the upper Hessenberg matrix H.

On exit, if INFO = 0 and WANTT is .TRUE., then H

contains the upper triangular matrix T from the Schur

decomposition (the Schur form). If INFO = 0 and WANT is

.FALSE., then the contents of H are unspecified on exit.

(The output value of H when INFO.GT.0 is given under the description of INFO below.)

This subroutine may explicitly set $H(i,j) = 0$ for $i.GT.j$ and $j = 1, 2, \dots, ILO-1$ or $j = IHI+1, IHI+2, \dots, N$.

[in] LDH

LDH is INTEGER

The leading dimension of the array H. LDH .GE. max(1,N).

[out] W

W is COMPLEX*16 array, dimension (N)

The computed eigenvalues of H(ILO:IHI,ILO:IHI) are stored in W(ILO:IHI). If WANTT is .TRUE., then the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with $W(i) = H(i,i)$.

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER

Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE..

1 .LE. ILOZ .LE. ILO; IHI .LE. IHIZ .LE. N.

[in,out] Z

Z is COMPLEX*16 array, dimension (LDZ,IHI)

If WANTZ is .FALSE., then Z is not referenced.

If WANTZ is .TRUE., then Z(ILO:IHI,ILOZ:IHIZ) is replaced by Z(ILO:IHI,ILOZ:IHIZ)*U where U is the orthogonal Schur factor of H(ILO:IHI,ILO:IHI).

(The output value of Z when INFO.GT.0 is given under the description of INFO below.)

[in] LDZ

LDZ is INTEGER

The leading dimension of the array Z. if WANTZ is .TRUE. then LDZ.GE.MAX(1,IHIZ). Otherwise, LDZ.GE.1.

[out] WORK

WORK is COMPLEX*16 array, dimension LWORK

On exit, if LWORK = -1, WORK(1) returns an estimate of the optimal value for LWORK.

[in] LWORK

LWORK is INTEGER

The dimension of the array WORK. LWORK .GE. max(1,N) is sufficient, but LWORK typically as large as 6*N may be required for optimal performance. A workspace query to determine the optimal workspace size is recommended.

If LWORK = -1, then ZLAQR4 does a workspace query.

In this case, ZLAQR4 checks the input parameters and estimates the optimal workspace size for the given values of N, ILO and IHI. The estimate is returned in WORK(1). No error message related to LWORK is issued by XERBLA. Neither H nor Z are accessed.

[out] INFO

INFO is INTEGER

= 0: successful exit

.GT. 0: if INFO = i, ZLAQR4 failed to compute all of the eigenvalues. Elements 1:ilo-1 and i+1:n of WR and WI contain those eigenvalues which have been successfully computed. (Failures are rare.)

If INFO .GT. 0 and WANT is .FALSE., then on exit,
the remaining unconverged eigenvalues are the eigen-
values of the upper Hessenberg matrix rows and
columns ILO through INFO of the final, output
value of H.

If INFO .GT. 0 and WANTT is .TRUE., then on exit

(*) (initial value of H)*U = U*(final value of H)

where U is a unitary matrix. The final
value of H is upper Hessenberg and triangular in
rows and columns INFO+1 through IHI.

If INFO .GT. 0 and WANTZ is .TRUE., then on exit

(final value of Z(ILO:IHI,ILOZ:IHIZ))
= (initial value of Z(ILO:IHI,ILOZ:IHIZ))*U

where U is the unitary matrix in (*) (regard-
less of the value of WANTT.)

If INFO .GT. 0 and WANTZ is .FALSE., then Z is not
accessed.

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Contributors:

=====

Karen Braman and Ralph Byers, Department of Mathematics,
University of Kansas, USA

References:

=====

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR
Algorithm Part I: Maintaining Well Focused Shifts, and Level 3
Performance, SIAM Journal of Matrix Analysis, volume 23, pages
929--947, 2002.

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR Algorithm Part II: Aggressive Early Deflation, SIAM Journal of Matrix Analysis, volume 23, pages 948--973, 2002.

— zlaqr4.f —

```
* =====
*      SUBROUTINE ZLAQR4( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
*      $                  IHIZ, Z, LDZ, WORK, LWORK, INFO )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, IHIZ, ILO, ILOZ, INFO, LDH, LDZ, LWORK, N
*      LOGICAL          WANTT, WANTZ
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
*      ..
*
* =====
*
*      .. Parameters ..
*
*      ==== Matrices of order NTINY or smaller must be processed by
*      .    ZLAHQR because of insufficient subdiagonal scratch space.
*      .    (This is a hard limit.) ====
*      INTEGER          NTINY
*      PARAMETER        ( NTINY = 11 )
*
*      ==== Exceptional deflation windows: try to cure rare
*      .    slow convergence by varying the size of the
*      .    deflation window after KEXNW iterations. ====
*      INTEGER          KEXNW
*      PARAMETER        ( KEXNW = 5 )
*
*      ==== Exceptional shifts: try to cure rare slow convergence
*      .    with ad-hoc exceptional shifts every KEXSH iterations.
*      .    ====
*      INTEGER          KEXSH
*      PARAMETER        ( KEXSH = 6 )
*
*      ==== The constant WILK1 is used to form the exceptional
```

```

*      .      shifts. ====
      DOUBLE PRECISION  WILK1
      PARAMETER          ( WILK1 = 0.75d0 )
      COMPLEX*16         ZERO, ONE
      PARAMETER          ( ZERO = ( 0.0d0, 0.0d0 ),
$      ONE = ( 1.0d0, 0.0d0 ) )
      DOUBLE PRECISION  TWO
      PARAMETER          ( TWO = 2.0d0 )

*      ..
*      .. Local Scalars ..
      COMPLEX*16         AA, BB, CC, CDUM, DD, DET, RTDISC, SWAP, TR2
      DOUBLE PRECISION  S
      INTEGER            I, INF, IT, ITMAX, K, KACC22, KBOT, KDU, KS,
$      KT, KTOP, KU, KV, KWH, KWTOP, KWV, LD, LS,
$      LWKOPT, NDEC, NDFL, NH, NHO, NIBBLE, NMIN, NS,
$      NSMAX, NSR, NVE, NW, NWMAX, NWR, NWUPBD
      LOGICAL            SORTED
      CHARACTER          JBCMPZ*2

*      ..
*      .. External Functions ..
      INTEGER            ILAENV
      EXTERNAL           ILAENV

*      ..
*      .. Local Arrays ..
      COMPLEX*16         ZDUM( 1, 1 )

*      ..
*      .. External Subroutines ..
      EXTERNAL           ZLACPY, ZLAHQR, ZLAQR2, ZLAQR5

*      ..
*      .. Intrinsic Functions ..
      INTRINSIC          ABS, DBLE, DCMPLX, DIMAG, INT, MAX, MIN, MOD,
$      SQRT

*      ..
*      .. Statement Functions ..
      DOUBLE PRECISION  CABS1

*      ..
*      .. Statement Function definitions ..
      CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )

*      ..
*      .. Executable Statements ..
      INFO = 0

*
*      ==== Quick return for N = 0: nothing to do. ====
*
      IF( N.EQ.0 ) THEN
        WORK( 1 ) = ONE
        RETURN
      END IF

*
      IF( N.LE.NTINY ) THEN

```

```

*
*      ==== Tiny matrices must use ZLAHQR. ====
*
      LWKOPT = 1
      IF( LWORK.NE.-1 )
$        CALL ZLAHQR( WANTT, WANTZ, N, ILO, IHI, H, LDH, W, ILOZ,
$                    IHIZ, Z, LDZ, INFO )
      ELSE
*
*      ==== Use small bulge multi-shift QR with aggressive early
*      .      deflation on larger-than-tiny matrices. ====
*
*      ==== Hope for the best. ====
*
      INFO = 0
*
*      ==== Set up job flags for ILAENV. ====
*
      IF( WANTT ) THEN
        JBCMPZ( 1: 1 ) = 'S'
      ELSE
        JBCMPZ( 1: 1 ) = 'E'
      END IF
      IF( WANTZ ) THEN
        JBCMPZ( 2: 2 ) = 'V'
      ELSE
        JBCMPZ( 2: 2 ) = 'N'
      END IF
*
*      ==== NWR = recommended deflation window size. At this
*      .      point, N .GT. NTINY = 11, so there is enough
*      .      subdiagonal workspace for NWR.GE.2 as required.
*      .      (In fact, there is enough subdiagonal space for
*      .      NWR.GE.3.) ====
*
      NWR = ILAENV( 13, 'ZLAQR4', JBCMPZ, N, ILO, IHI, LWORK )
      NWR = MAX( 2, NWR )
      NWR = MIN( IHI-ILO+1, ( N-1 ) / 3, NWR )
*
*      ==== NSR = recommended number of simultaneous shifts.
*      .      At this point N .GT. NTINY = 11, so there is at
*      .      enough subdiagonal workspace for NSR to be even
*      .      and greater than or equal to two as required. ====
*
      NSR = ILAENV( 15, 'ZLAQR4', JBCMPZ, N, ILO, IHI, LWORK )
      NSR = MIN( NSR, ( N+6 ) / 9, IHI-ILO )
      NSR = MAX( 2, NSR-MOD( NSR, 2 ) )
*
*      ==== Estimate optimal workspace ====
*

```

```

*      ==== Workspace query call to ZLAQR2 ====
*
      CALL ZLAQR2( WANTT, WANTZ, N, ILO, IHI, NWR+1, H, LDH, ILOZ,
$              IHIZ, Z, LDZ, LS, LD, W, H, LDH, N, H, LDH, N, H,
$              LDH, WORK, -1 )
*
*      ==== Optimal workspace = MAX(ZLAQR5, ZLAQR2) ====
*
      LWKOPT = MAX( 3*NSR / 2, INT( WORK( 1 ) ) )
*
*      ==== Quick return in case of workspace query. ====
*
      IF( LWORK.EQ.-1 ) THEN
          WORK( 1 ) = DCMPLX( LWKOPT, 0 )
          RETURN
      END IF
*
*      ==== ZLAHQR/ZLAQRO crossover point ====
*
      NMIN = ILAENV( 12, 'ZLAQR4', JBCMPZ, N, ILO, IHI, LWORK )
      NMIN = MAX( NTINY, NMIN )
*
*      ==== Nibble crossover point ====
*
      NIBBLE = ILAENV( 14, 'ZLAQR4', JBCMPZ, N, ILO, IHI, LWORK )
      NIBBLE = MAX( 0, NIBBLE )
*
*      ==== Accumulate reflections during ttswp? Use block
*      .      2-by-2 structure during matrix-matrix multiply? ====
*
      KACC22 = ILAENV( 16, 'ZLAQR4', JBCMPZ, N, ILO, IHI, LWORK )
      KACC22 = MAX( 0, KACC22 )
      KACC22 = MIN( 2, KACC22 )
*
*      ==== NWMAX = the largest possible deflation window for
*      .      which there is sufficient workspace. ====
*
      NWMAX = MIN( ( N-1 ) / 3, LWORK / 2 )
      NW = NWMAX
*
*      ==== NSMAX = the Largest number of simultaneous shifts
*      .      for which there is sufficient workspace. ====
*
      NSMAX = MIN( ( N+6 ) / 9, 2*LWORK / 3 )
      NSMAX = NSMAX - MOD( NSMAX, 2 )
*
*      ==== NDFL: an iteration count restarted at deflation. ====
*
      NDFL = 1
*

```

```

*      ==== ITMAX = iteration limit ====
*
*      ITMAX = MAX( 30, 2*KEXSH )*MAX( 10, ( IHI-ILO+1 ) )
*
*      ==== Last row and column in the active block ====
*
*      KBOT = IHI
*
*      ==== Main Loop ====
*
*      DO 70 IT = 1, ITMAX
*
*          ==== Done when KBOT falls below ILO ====
*
*          IF( KBOT.LT.ILO )
*      $              GO TO 80
*
*          ==== Locate active block ====
*
*          DO 10 K = KBOT, ILO + 1, -1
*              IF( H( K, K-1 ).EQ.ZERO )
*      $                  GO TO 20
*      10      CONTINUE
*              K = ILO
*      20      CONTINUE
*              KTOP = K
*
*          ==== Select deflation window size:
*          .   Typical Case:
*          .       If possible and advisable, nibble the entire
*          .       active block.  If not, use size MIN(NWR,NWMAX)
*          .       or MIN(NWR+1,NWMAX) depending upon which has
*          .       the smaller corresponding subdiagonal entry
*          .       (a heuristic).
*          .
*          .   Exceptional Case:
*          .       If there have been no deflations in KEXNW or
*          .       more iterations, then vary the deflation window
*          .       size.  At first, because, larger windows are,
*          .       in general, more powerful than smaller ones,
*          .       rapidly increase the window to the maximum possible.
*          .       Then, gradually reduce the window size. ====
*
*          NH = KBOT - KTOP + 1
*          NWUPBD = MIN( NH, NWMAX )
*          IF( NDFL.LT.KEXNW ) THEN
*              NW = MIN( NWUPBD, NWR )
*          ELSE
*              NW = MIN( NWUPBD, 2*NW )
*          END IF

```



```

      IF( NW.LT.NWMAX ) THEN
        IF( NW.GE.NH-1 ) THEN
          NW = NH
        ELSE
          KWTOP = KBOT - NW + 1
          IF( CABS1( H( KWTOP, KWTOP-1 ) ).GT.
$           CABS1( H( KWTOP-1, KWTOP-2 ) ) )NW = NW + 1
        END IF
      END IF
      IF( NDFL.LT.KEXNW ) THEN
        NDEC = -1
      ELSE IF( NDEC.GE.0 .OR. NW.GE.NWUPBD ) THEN
        NDEC = NDEC + 1
        IF( NW-NDEC.LT.2 )
$         NDEC = 0
        NW = NW - NDEC
      END IF

*
*      ==== Aggressive early deflation:
*      .      split workspace under the subdiagonal into
*      .      - an nw-by-nw work array V in the lower
*      .      left-hand-corner,
*      .      - an NW-by-at-least-NW-but-more-is-better
*      .      (NW-by-NHO) horizontal work array along
*      .      the bottom edge,
*      .      - an at-least-NW-but-more-is-better (NHV-by-NW)
*      .      vertical work array along the left-hand-edge.
*      .      ====
*
      KV = N - NW + 1
      KT = NW + 1
      NHO = ( N-NW-1 ) - KT + 1
      KWV = NW + 2
      NVE = ( N-NW ) - KWV + 1

*
*      ==== Aggressive early deflation ====
*
$      CALL ZLAQR2( WANTT, WANTZ, N, KTOP, KBOT, NW, H, LDH, ILOZ,
$                  IHIZ, Z, LDZ, LS, LD, W, H( KV, 1 ), LDH, NHO,
$                  H( KV, KT ), LDH, NVE, H( KWV, 1 ), LDH, WORK,
$                  LWORK )

*
*      ==== Adjust KBOT accounting for new deflations. ====
*
      KBOT = KBOT - LD

*
*      ==== KS points to the shifts. ====
*
      KS = KBOT - LS + 1
*

```

```

*      ==== Skip an expensive QR sweep if there is a (partly
*      .      heuristic) reason to expect that many eigenvalues
*      .      will deflate without it. Here, the QR sweep is
*      .      skipped if many eigenvalues have just been deflated
*      .      or if the remaining active block is small.
*
*      IF( ( LD.EQ.0 ) .OR. ( ( 100*LD.LE.NW*NIBBLE ) .AND. ( KBOT-
$      KTOP+1.GT.MIN( NMIN, NWMAX ) ) ) ) THEN
*
*      ==== NS = nominal number of simultaneous shifts.
*      .      This may be lowered (slightly) if ZLAQR2
*      .      did not provide that many shifts. ====
*
*      NS = MIN( NSMAX, NSR, MAX( 2, KBOT-KTOP ) )
*      NS = NS - MOD( NS, 2 )
*
*      ==== If there have been no deflations
*      .      in a multiple of KEXSH iterations,
*      .      then try exceptional shifts.
*      .      Otherwise use shifts provided by
*      .      ZLAQR2 above or from the eigenvalues
*      .      of a trailing principal submatrix. ====
*
*      IF( MOD( NDFL, KEXSH ).EQ.0 ) THEN
*      KS = KBOT - NS + 1
*      DO 30 I = KBOT, KS + 1, -2
*          W( I ) = H( I, I ) + WILK1*CABS1( H( I, I-1 ) )
*          W( I-1 ) = W( I )
30      CONTINUE
*      ELSE
*
*      ==== Got NS/2 or fewer shifts? Use ZLAHQR
*      .      on a trailing principal submatrix to
*      .      get more. (Since NS.LE.NSMAX.LE.(N+6)/9,
*      .      there is enough space below the subdiagonal
*      .      to fit an NS-by-NS scratch array.) ====
*
*      IF( KBOT-KS+1.LE.NS / 2 ) THEN
*      KS = KBOT - NS + 1
*      KT = N - NS + 1
*      CALL ZLACPY( 'A', NS, NS, H( KS, KS ), LDH,
$      H( KT, 1 ), LDH )
*      CALL ZLAHQR( .false., .false., NS, 1, NS,
$      H( KT, 1 ), LDH, W( KS ), 1, 1, ZDUM,
$      1, INF )
*      KS = KS + INF
*
*      ==== In case of a rare QR failure use
*      .      eigenvalues of the trailing 2-by-2
*      .      principal submatrix. Scale to avoid

```

```

*           .   overflows, underflows and subnormals.
*           .   (The scale factor S can not be zero,
*           .   because H(KBOT,KBOT-1) is nonzero.) ====
*
      IF( KS.GE.KBOT ) THEN
        S = CABS1( H( KBOT-1, KBOT-1 ) ) +
$         CABS1( H( KBOT, KBOT-1 ) ) +
$         CABS1( H( KBOT-1, KBOT ) ) +
$         CABS1( H( KBOT, KBOT ) )
        AA = H( KBOT-1, KBOT-1 ) / S
        CC = H( KBOT, KBOT-1 ) / S
        BB = H( KBOT-1, KBOT ) / S
        DD = H( KBOT, KBOT ) / S
        TR2 = ( AA+DD ) / TWO
        DET = ( AA-TR2 )*( DD-TR2 ) - BB*CC
        RTDISC = SQRT( -DET )
        W( KBOT-1 ) = ( TR2+RTDISC )*S
        W( KBOT ) = ( TR2-RTDISC )*S
*
        KS = KBOT - 1
      END IF
    END IF
*
    IF( KBOT-KS+1.GT.NS ) THEN
*
*       ==== Sort the shifts (Helps a little) ====
*
      SORTED = .false.
      DO 50 K = KBOT, KS + 1, -1
        IF( SORTED )
$          GO TO 60
        SORTED = .true.
        DO 40 I = KS, K - 1
          IF( CABS1( W( I ) ).LT.CABS1( W( I+1 ) ) )
$            THEN
              SORTED = .false.
              SWAP = W( I )
              W( I ) = W( I+1 )
              W( I+1 ) = SWAP
            END IF
          CONTINUE
40        CONTINUE
50        CONTINUE
60        CONTINUE
      END IF
    END IF
*
*       ==== If there are only two shifts, then use
*       .   only one. ====
*
    IF( KBOT-KS+1.EQ.2 ) THEN

```

```

      IF( CABS1( W( KBOT )-H( KBOT, KBOT ) ).LT.
$      CABS1( W( KBOT-1 )-H( KBOT, KBOT ) ) ) THEN
      W( KBOT-1 ) = W( KBOT )
      ELSE
      W( KBOT ) = W( KBOT-1 )
      END IF
END IF

*
*      ==== Use up to NS of the the smallest magnatiude
*      .      shifts.  If there aren't NS shifts available,
*      .      then use them all, possibly dropping one to
*      .      make the number of shifts even. ====
*
      NS = MIN( NS, KBOT-KS+1 )
      NS = NS - MOD( NS, 2 )
      KS = KBOT - NS + 1

*
*      ==== Small-bulge multi-shift QR sweep:
*      .      split workspace under the subdiagonal into
*      .      - a KDU-by-KDU work array U in the lower
*      .      left-hand-corner,
*      .      - a KDU-by-at-least-KDU-but-more-is-better
*      .      (KDU-by-NHo) horizontal work array WH along
*      .      the bottom edge,
*      .      - and an at-least-KDU-but-more-is-better-by-KDU
*      .      (NVE-by-KDU) vertical work WV arrow along
*      .      the left-hand-edge. ====
*
      KDU = 3*NS - 3
      KU = N - KDU + 1
      KWH = KDU + 1
      NHO = ( N-KDU+1-4 ) - ( KDU+1 ) + 1
      KWV = KDU + 4
      NVE = N - KDU - KWV + 1

*
*      ==== Small-bulge multi-shift QR sweep ====
*
      CALL ZLAQR5( WANTT, WANTZ, KACC22, N, KTOP, KBOT, NS,
$      W( KS ), H, LDH, ILOZ, IHIZ, Z, LDZ, WORK,
$      3, H( KU, 1 ), LDH, NVE, H( KWV, 1 ), LDH,
$      NHO, H( KU, KWH ), LDH )
END IF

*
*      ==== Note progress (or the lack of it). ====
*
      IF( LD.GT.0 ) THEN
      NDFL = 1
      ELSE
      NDFL = NDFL + 1
      END IF

```

```

*
*      ==== End of main loop ====
70    CONTINUE
*
*      ==== Iteration limit exceeded. Set INFO to show where
*      .    the problem occurred and exit. ====
*
*      INFO = KBOT
80    CONTINUE
*      END IF
*
*      ==== Return the optimal value of LWORK. ====
*
*      WORK( 1 ) = DCMPLX( LWKOPT, 0 )
*
*      ==== End of ZLAQR4 ====
*
*      END

```

— LAPACK zlaqr4 —

```

(let*
  ((ntiny 11) (kexnw 5) (kexsh 6) (wilk1 0.75d0)
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (two 2.0d0))
  (declare (type (f2cl-lib:integer4 11 11) ntiny)
            (type (f2cl-lib:integer4 5 5) kexnw) (type (f2cl-lib:integer4 6 6) kexsh)
            (type (double-float 0.75d0 0.75d0) wilk1) (type (f2cl-lib:complex16) zero)
            (type (f2cl-lib:complex16) one) (type (double-float 2.0d0 2.0d0) two)
            (ignorable ntiny kexnw kexsh wilk1 zero one two))
  (defun zlaqr4 (wantt wantz n ilo ihi h ldh w iloz ihiz z ldz work lwork info)
    (declare (type f2cl-lib:logical wantt wantz)
              (type (f2cl-lib:integer4) info lwork ldz ihiz iloz ldh ihi ilo n)
              (type (array f2cl-lib:complex16 (*)) work z w h))
    (f2cl-lib:with-multi-array-data
      ((h f2cl-lib:complex16 h-%data% h-%offset%)
       (w f2cl-lib:complex16 w-%data% w-%offset%)
       (z f2cl-lib:complex16 z-%data% z-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (labels
        ((cabs1 (cdum)
          (+ (abs (f2cl-lib:dble cdum)) (abs (f2cl-lib:dimag cdum))))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                           (values double-float &rest t)) cabs1))
        (prog

```

```

((zsum (make-array 1 :element-type 'f2cl-lib:complex16))
 (jbcmpz (make-array '(2) :element-type 'character
                    :initial-element #\space))
 (sorted nil) (i 0) (inf 0) (it 0) (itmax 0) (k 0) (kacc22 0) (kbot 0)
 (kdu 0) (ks 0) (kt 0) (ktop 0) (ku 0) (kv 0) (kwh 0) (kwttop 0) (kwv 0)
 (ld 0) (ls 0) (lwkopt 0) (ndec 0) (ndfl 0) (nh 0) (nho 0) (nibble 0)
 (nmin 0) (ns 0) (nsmax 0) (nsr 0) (nve 0) (nw 0) (nwmax 0) (nwr 0)
 (nwupbd 0) (s 0.0d0) (aa #C(0.0d0 0.0d0)) (bb #C(0.0d0 0.0d0))
 (cc #C(0.0d0 0.0d0)) (cdum #C(0.0d0 0.0d0)) (dd #C(0.0d0 0.0d0))
 (det #C(0.0d0 0.0d0)) (rtdisc #C(0.0d0 0.0d0)) (swap #C(0.0d0 0.0d0))
 (tr2 #C(0.0d0 0.0d0)))
(declare (type (array f2cl-lib:complex16 (1)) zsum)
 (type (simple-array character (2)) jbcmpz)
 (type f2cl-lib:logical sorted)
 (type (f2cl-lib:integer4) nwupbd nwr nwmax nw nve nsr nsmax
 ns nmin nibble
 nho nh ndfl ndec lwkopt ls ld kwv kwttop kwh kv ku ktop kt
 ks kdu kbot
 kacc22 k itmax it inf i)
 (type (double-float) s)
 (type (f2cl-lib:complex16) tr2 swap rtdisc det dd cdum cc bb aa))
(setf info 0)
(cond
 ((= n 0)
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%) one)
  (go end_label)))
 (cond
 ((<= n ntiny) (setf lwkopt 1)
 (if (/= lwork -1)
  (multiple-value-bind
   (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
    var-8 var-9 var-10
    var-11 var-12)
   (zlahqr wantt wantz n ilo ihi h ldh w iloz ihiz z ldz info)
  (declare
   (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-7
    var-8 var-9 var-10
    var-11))
  (setf ldh var-6) (setf info var-12))))))
 (t
  (tagbody (setf info 0)
   (cond
    (wantt
     (f2cl-lib:fset-string (f2cl-lib:fref-string jbcmpz (1 1)) "S"))
    (t
     (f2cl-lib:fset-string (f2cl-lib:fref-string jbcmpz (1 1)) "E")))
   (cond
    (wantz
     (f2cl-lib:fset-string (f2cl-lib:fref-string jbcmpz (2 2)) "V"))
    (t

```

```

(f2cl-lib:fset-string (f2cl-lib:fref-string jbcmpz (2 2)) "N"))
(setf nwr
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 13 "ZLAQR4" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nwr (max (the f2cl-lib:integer4 2)
  (the f2cl-lib:integer4 nwr)))
(setf nwr
  (min (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1)
    (the f2cl-lib:integer4 (truncate (- n 1) 3)) nwr))
(setf nsr
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 15 "ZLAQR4" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nsr
  (min nsr (the f2cl-lib:integer4 (truncate (+ n 6) 9))
    (f2cl-lib:int-sub ihi ilo)))
(setf nsr
  (max (the f2cl-lib:integer4 2)
    (the f2cl-lib:integer4 (f2cl-lib:int-sub nsr (mod nsr 2)))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10
   var-11 var-12 var-13 var-14 var-15 var-16 var-17
   var-18 var-19 var-20
   var-21 var-22 var-23 var-24)
  (zlaqr2 wantt wantz n ilo ihi
    (f2cl-lib:int-add nwr 1) h ldh iloz ihiz z
    ldz ls ld w h ldh n h ldh n h ldh work -1)
  (declare
    (ignore var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-8
      var-9 var-10
      var-14 var-15 var-17 var-18 var-20 var-21 var-23 var-24))
  (setf ldh var-7)
  (setf ldz var-11)
  (setf ls var-12)
  (setf ld var-13)
  (setf ldh var-16)
  (setf ldh var-19)
  (setf ldh var-22))
(setf lwkopt
  (max (the f2cl-lib:integer4 (truncate (* 3 nsr) 2))

```

```

(f2cl-lib:int
  (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)))
(cond
  ((= lwork (f2cl-lib:int-sub 1))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (f2cl-lib:dcmplx lwkopt 0))
    (go end_label)))
(setf nmin
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 12 "ZLAQR4" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nmin
  (max (the f2cl-lib:integer4 ntiny) (the f2cl-lib:integer4 nmin)))
(setf nibble
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 14 "ZLAQR4" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf nibble
  (max (the f2cl-lib:integer4 0) (the f2cl-lib:integer4 nibble)))
(setf kacc22
  (multiple-value-bind
    (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
    (ilaenv 16 "ZLAQR4" jbcmpz n ilo ihi lwork)
    (declare (ignore var-0 var-1)) (when var-2 (setf jbcmpz var-2))
    (when var-3 (setf n var-3)) (when var-4 (setf ilo var-4))
    (when var-5 (setf ihi var-5))
    (when var-6 (setf lwork var-6)) ret-val))
(setf kacc22
  (max (the f2cl-lib:integer4 0) (the f2cl-lib:integer4 kacc22)))
(setf kacc22
  (min (the f2cl-lib:integer4 2) (the f2cl-lib:integer4 kacc22)))
(setf nwmax
  (min (the f2cl-lib:integer4 (truncate (- n 1) 3))
        (the f2cl-lib:integer4 (truncate lwork 2))))
(setf nw nwmax)
(setf nsmax
  (min (the f2cl-lib:integer4 (truncate (+ n 6) 9))
        (the f2cl-lib:integer4 (truncate (* 2 lwork) 3))))
(setf nsmax (f2cl-lib:int-sub nsmax (mod nsmax 2))) (setf ndfl 1)
(setf itmax
  (f2cl-lib:int-mul
    (max (the f2cl-lib:integer4 30)

```



```

      (the f2cl-lib:integer4 (f2cl-lib:int-mul 2 kexsh)))
      (max (the f2cl-lib:integer4 10)
      (the f2cl-lib:integer4
      (f2cl-lib:int-add (f2cl-lib:int-sub ihi ilo) 1))))))
      (setf kbot ihi)
      (f2cl-lib:fdo (it 1 (f2cl-lib:int-add it 1))
      ((> it itmax) nil)
      (tagbody
      (if (< kbot ilo) (go label80))
      (f2cl-lib:fdo (k kbot
      (f2cl-lib:int-add k (f2cl-lib:int-sub 1)))
      ((> k
      (f2cl-lib:int-add ilo 1))
      nil)
      (tagbody
      (if
      (=
      (f2cl-lib:fref h-%data% (k
      (f2cl-lib:int-sub k 1)) ((1 ldh) (1 *))
      h-%offset%)
      zero)
      (go label20))
      label10))
      (setf k ilo) label20 (setf ktop k)
      (setf nh (f2cl-lib:int-add
      (f2cl-lib:int-sub kbot ktop) 1))
      (setf nwupbd
      (min (the f2cl-lib:integer4 nh)
      (the f2cl-lib:integer4 nwmax)))
      (cond
      ((< ndfl kexnw)
      (setf nw
      (min (the f2cl-lib:integer4 nwupbd)
      (the f2cl-lib:integer4 nwr))))
      (t
      (setf nw
      (min (the f2cl-lib:integer4 nwupbd)
      (the f2cl-lib:integer4
      (f2cl-lib:int-mul 2 nw))))))
      (cond
      ((< nw nwmax)
      (cond ((>= nw (f2cl-lib:int-add nh
      (f2cl-lib:int-sub 1)))
      (setf nw nh))
      (t (setf kwtop (f2cl-lib:int-add
      (f2cl-lib:int-sub kbot nw) 1))
      (if
      (>
      (cabs1
      (f2cl-lib:fref h-%data% (kwtop

```

```

(f2cl-lib:int-sub kwttop 1))
((1 ldh) (1 *)) h-%offset%)
(cabs1
(f2cl-lib:fref h-%data%
((f2cl-lib:int-sub kwttop 1)
(f2cl-lib:int-sub kwttop 2))
((1 ldh) (1 *)) h-%offset%)))
(setf nw (f2cl-lib:int-add nw 1))))))
(cond ((< ndfl kexnw) (setf ndec -1))
((or (>= ndec 0) (>= nw nwupbd))
(setf ndec (f2cl-lib:int-add ndec 1))
(if (< (f2cl-lib:int-sub nw ndec) 2) (setf ndec 0))
(setf nw (f2cl-lib:int-sub nw ndec))))
(setf kv (f2cl-lib:int-add
(f2cl-lib:int-sub n nw) 1))
(setf kt (f2cl-lib:int-add nw 1))
(setf nho (f2cl-lib:int-add
(f2cl-lib:int-sub n nw 1 kt) 1))
(setf kwv (f2cl-lib:int-add nw 2))
(setf nve (f2cl-lib:int-add
(f2cl-lib:int-sub n nw kwv) 1))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6
var-7 var-8 var-9 var-10
var-11 var-12 var-13 var-14 var-15 var-16
var-17 var-18 var-19 var-20
var-21 var-22 var-23 var-24)
(zlaqr2 wantt wantz n ktop kbot nw h ldh
iloz ihiz z ldz ls ld w
(f2cl-lib:array-slice h-%data%
f2cl-lib:complex16 (kv 1)
((1 ldh) (1 *)) h-%offset%)
ldh nho
(f2cl-lib:array-slice h-%data%
f2cl-lib:complex16 (kv kt)
((1 ldh) (1 *)) h-%offset%)
ldh nve
(f2cl-lib:array-slice h-%data%
f2cl-lib:complex16 (kwv 1)
((1 ldh) (1 *)) h-%offset%)
ldh work lwork)
(declare
(ignore var-0 var-1 var-2 var-3 var-4 var-5
var-6 var-8 var-9 var-10
var-14 var-15 var-17 var-18 var-20 var-21
var-23 var-24))
(setf ldh var-7)
(setf ldz var-11)
(setf ls var-12)
(setf ld var-13)

```

```

(setf ldh var-16)
(setf ldh var-19)
(setf ldh var-22))
(setf kbot (f2cl-lib:int-sub kbot ld))
(setf ks (f2cl-lib:int-add
          (f2cl-lib:int-sub kbot ls) 1))
(cond
  ((or (= ld 0)
        (and (<= (f2cl-lib:int-mul 100 ld)
                  (f2cl-lib:int-mul nw nibble))
              (> (f2cl-lib:int-add kbot
                                   (f2cl-lib:int-sub ktop) 1)
                 (min (the f2cl-lib:integer4 nmin)
                       (the f2cl-lib:integer4 nwmax))))))
    (setf ns
      (min (the f2cl-lib:integer4 nsmax)
            (the f2cl-lib:integer4 nsr)
            (the f2cl-lib:integer4
              (max (the f2cl-lib:integer4 2)
                    (the f2cl-lib:integer4
                      (f2cl-lib:int-sub kbot ktop)))))))
    (setf ns (f2cl-lib:int-sub ns (mod ns 2)))
    (cond
      ((= (mod ndfl kexsh) 0)
       (setf ks (f2cl-lib:int-add
                  (f2cl-lib:int-sub kbot ns) 1))
       (f2cl-lib:fdo (i kbot (f2cl-lib:int-add i
                                                  (f2cl-lib:int-sub 2)))
                     ((> i
                          (f2cl-lib:int-add ks 1))
                      nil)
       (tagbody
         (setf (f2cl-lib:fref w-%data% (i)
                              ((1 *)) w-%offset%)
               (+ (f2cl-lib:fref h-%data% (i i)
                                   ((1 ldh) (1 *)) h-%offset%)
                  (* wilk1
                     (cabs1
                      (f2cl-lib:fref h-%data% (i
                                                (f2cl-lib:int-sub i 1))
                                   ((1 ldh) (1 *)) h-%offset%))))))
         (setf
          (f2cl-lib:fref w-%data%
                        ((f2cl-lib:int-sub i 1)) ((1 *))
            w-%offset%)
          (f2cl-lib:fref w-%data% (i) ((1 *))
                        w-%offset%))
         label30)))
    (t
     (cond

```

```

((<= (f2cl-lib:int-add kbot
      (f2cl-lib:int-sub ks) 1)
  (f2cl-lib:f2cl/ ns 2))
(setf ks (f2cl-lib:int-add
          (f2cl-lib:int-sub kbot ns) 1))
(setf kt (f2cl-lib:int-add
          (f2cl-lib:int-sub n ns) 1))
(zlacpy "A" ns ns
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16 (ks ks)
    ((1 ldh) (1 *)) h-%offset%)
  ldh
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16 (kt 1)
    ((1 ldh) (1 *)) h-%offset%)
  ldh)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5
   var-6 var-7 var-8 var-9
   var-10 var-11 var-12)
  (zlahqr f2cl-lib:%false% f2cl-lib:%false%
    ns 1 ns
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (kt 1)
      ((1 ldh) (1 *)) h-%offset%)
    ldh
    (f2cl-lib:array-slice w-%data%
      f2cl-lib:complex16 (ks) ((1 *))
      w-%offset%)
    1 1 zdum 1 inf)
  (declare
    (ignore var-0 var-1 var-2 var-3 var-4
      var-5 var-7 var-8 var-9
      var-10 var-11))
    (setf ldh var-6) (setf inf var-12))
  (setf ks (f2cl-lib:int-add ks inf))
  (cond
    ((>= ks kbot)
     (setf s
      (+
        (cabs1
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-sub kbot 1)
              (f2cl-lib:int-sub kbot 1))
            ((1 ldh) (1 *)) h-%offset%))
        (cabs1
          (f2cl-lib:fref h-%data% (kbot
            (f2cl-lib:int-sub kbot 1))
            ((1 ldh) (1 *)) h-%offset%))
        (cabs1

```

```

(f2cl-lib:fref h-%data%
  ((f2cl-lib:int-sub kbot 1) kbot)
  ((1 ldh) (1 *)) h-%offset%))
(cabs1
  (f2cl-lib:fref h-%data% (kbot kbot)
    ((1 ldh) (1 *))
    h-%offset%))))
(setf aa
  (/
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-sub kbot 1)
        (f2cl-lib:int-sub kbot 1))
      ((1 ldh) (1 *)) h-%offset%
    s))
  (setf cc
    (/
      (f2cl-lib:fref h-%data% (kbot
        (f2cl-lib:int-sub kbot 1))
        ((1 ldh) (1 *)) h-%offset%
      s))
    (setf bb
      (/
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-sub kbot 1) kbot)
          ((1 ldh) (1 *)) h-%offset%
        s))
        (setf dd
          (/
            (f2cl-lib:fref h-%data% (kbot kbot)
              ((1 ldh) (1 *)) h-%offset%
            s))
            (setf tr2 (/ (+ aa dd) two))
            (setf det (- (* (- aa tr2) (- dd tr2))
              (* bb cc)))
            (setf rtdisc (f2cl-lib:fsqrt (- det)))
            (setf
              (f2cl-lib:fref w-%data%
                ((f2cl-lib:int-sub kbot 1) ((1 *))
                w-%offset%
              (* (+ tr2 rtdisc) s))
              (setf (f2cl-lib:fref w-%data% (kbot)
                ((1 *)) w-%offset%
              (* (- tr2 rtdisc) s))
              (setf ks (f2cl-lib:int-sub kbot 1))))))
            (cond
              ((> (f2cl-lib:int-add kbot
                (f2cl-lib:int-sub ks) 1) ns)
                (tagbody (setf sorted f2cl-lib:%false%)
                  (f2cl-lib:fdo (k kbot (f2cl-lib:int-add k
                    (f2cl-lib:int-sub 1)))

```

```

(<
      k (f2cl-lib:int-add ks 1))
      nil)
(tagbody (if sorted (go label60))
  (setf sorted f2cl-lib:%true%)
  (f2cl-lib:fdo (i ks
    (f2cl-lib:int-add i 1))
    ((> i
      (f2cl-lib:int-add k
        (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
      (cond
        ((< (cabs1 (f2cl-lib:fref w (i)
          ((1 *))))
          (cabs1 (f2cl-lib:fref w
            ((f2cl-lib:int-add i 1))
            ((1 *))))))
        (setf sorted f2cl-lib:%false%)
        (setf swap (f2cl-lib:fref
          w-%data% (i) ((1 *))
          w-%offset%))
        (setf (f2cl-lib:fref w-%data%
          (i) ((1 *)) w-%offset%)
          (f2cl-lib:fref w-%data%
            ((f2cl-lib:int-add i 1)) ((1 *))
            w-%offset%))
        (setf
          (f2cl-lib:fref w-%data%
            ((f2cl-lib:int-add i 1)) ((1 *))
            w-%offset%)
          swap)))
        label40))
      label50))
    label60))))
(cond
  ((= (f2cl-lib:int-add kbot
    (f2cl-lib:int-sub ks 1) 2)
    (cond
      ((<
        (cabs1
          (+ (f2cl-lib:fref w (kbot) ((1 *)))
            (- (f2cl-lib:fref h (kbot kbot)
              ((1 ldh) (1 *))))))
        (cabs1
          (+
            (f2cl-lib:fref w
              ((f2cl-lib:int-add kbot
                (f2cl-lib:int-sub 1)))
              ((1 *)))

```

```

        (- (f2cl-lib:fref h (kbot kbot)
              ((1 ldh) (1 *))))))
(setf
  (f2cl-lib:fref w-%data%
    ((f2cl-lib:int-sub kbot 1)) ((1 *))
    w-%offset%)
  (f2cl-lib:fref w-%data% (kbot) ((1 *))
    w-%offset%))
(t
  (setf (f2cl-lib:fref w-%data% (kbot)
    ((1 *)) w-%offset%)
    (f2cl-lib:fref w-%data%
      ((f2cl-lib:int-sub kbot 1)) ((1 *))
      w-%offset%))))))
(setf ns
  (min (the f2cl-lib:integer4 ns)
    (the f2cl-lib:integer4
      (f2cl-lib:int-add (f2cl-lib:int-sub kbot ks) 1))))
(setf ns (f2cl-lib:int-sub ns (mod ns 2)))
(setf ks (f2cl-lib:int-add
  (f2cl-lib:int-sub kbot ns) 1))
(setf kdu (f2cl-lib:int-sub
  (f2cl-lib:int-mul 3 ns) 3))
(setf ku (f2cl-lib:int-add
  (f2cl-lib:int-sub n kdu) 1))
(setf kwh (f2cl-lib:int-add kdu 1))
(setf nho
  (f2cl-lib:int-add
    (f2cl-lib:int-sub
      (f2cl-lib:int-add (f2cl-lib:int-sub n kdu) 1) 4
      (f2cl-lib:int-add kdu 1))
    1))
(setf kwv (f2cl-lib:int-add kdu 4))
(setf nve
  (f2cl-lib:int-add (f2cl-lib:int-sub n kdu kwv) 1))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6
   var-7 var-8 var-9 var-10
   var-11 var-12 var-13 var-14 var-15 var-16
   var-17 var-18 var-19 var-20
   var-21 var-22 var-23)
  (zlaqr5 wantt wantz kacc22 n ktop kbot ns
    (f2cl-lib:array-slice w-%data%
      f2cl-lib:complex16 (ks) ((1 *))
      w-%offset%)
    h ldh iloz ihiz z ldz work 3
    (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 (ku 1)
      ((1 ldh) (1 *)) h-%offset%)
    ldh nve

```

```

(f2cl-lib:array-slice h-%data%
 f2cl-lib:complex16 (kwv 1)
 ((1 ldh) (1 *)) h-%offset%)
ldh nho
(f2cl-lib:array-slice h-%data%
 f2cl-lib:complex16 (ku kwh)
 ((1 ldh) (1 *)) h-%offset%)
ldh)
(declare
 (ignore var-7 var-8 var-12 var-14 var-15
  var-16 var-19 var-22))
(when var-0 (setf wantt var-0))
(when var-1 (setf wantz var-1))
(when var-2 (setf kacc22 var-2))
(when var-3 (setf n var-3))
(when var-4 (setf ktop var-4))
(when var-5 (setf kbot var-5))
(when var-6 (setf ns var-6))
(when var-9 (setf ldh var-9))
(when var-10 (setf iloz var-10))
(when var-11 (setf ihiz var-11))
(when var-13 (setf ldz var-13))
(when var-17 (setf ldh var-17))
(when var-18 (setf nve var-18))
(when var-20 (setf ldh var-20))
(when var-21 (setf nho var-21))
(when var-23 (setf ldh var-23))))
(cond ((> ld 0) (setf ndfl 1))
      (t (setf ndfl (f2cl-lib:int-add ndfl 1))))
label70))
(setf info kbot) label80)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
 (f2cl-lib:dcmplx lwkopt 0))
end_label
(return
 (values wantt wantz n ilo ihi nil ldh nil iloz ihiz nil ldz nil lwork
  info))))))

```

zlaqr5 LAPACK

— zlaqr5.input —

```

)set break resume
)sys rm -f zlaqr5.output
)spool zlaqr5.output

```



```
)set message test on
)set message auto off
)clear all
```

```
)spool
)lisp (bye)
```

—————
— zlaqr5.help —

```
=====
zlaqr5 examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```
SUBROUTINE ZLAQR5( WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, S,
                  H, LDH, ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV,
                  WV, LDWV, NH, WH, LDWH )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHIZ, ILOZ, KACC22, KBOT, KTOP, LDH, LDU, LDV,
$                LDWH, LDWV, LDZ, N, NH, NSHFTS, NV
LOGICAL          WANTT, WANTZ
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       H( LDH, * ), S( * ), U( LDU, * ), V( LDV, * ),
$                WH( LDWH, * ), WV( LDWV, * ), Z( LDZ, * )
```

```
..
```

Purpose:

=====

ZLAQR5, called by ZLAQR0, performs a
single small-bulge multi-shift QR sweep.

Arguments:

=====

[in] WANTT

WANTT is logical scalar
WANTT = .true. if the triangular Schur factor
is being computed. WANTT is set to .false. otherwise.

[in] WANTZ

WANTZ is logical scalar
WANTZ = .true. if the unitary Schur factor is being
computed. WANTZ is set to .false. otherwise.

[in] KACC22

KACC22 is integer with value 0, 1, or 2.
Specifies the computation mode of far-from-diagonal
orthogonal updates.
= 0: ZLAQR5 does not accumulate reflections and does not
use matrix-matrix multiply to update far-from-diagonal
matrix entries.
= 1: ZLAQR5 accumulates reflections and uses matrix-matrix
multiply to update the far-from-diagonal matrix entries.
= 2: ZLAQR5 accumulates reflections, uses matrix-matrix
multiply to update the far-from-diagonal matrix entries,
and takes advantage of 2-by-2 block structure during
matrix multiplies.

[in] N

N is integer scalar
N is the order of the Hessenberg matrix H upon which this
subroutine operates.

[in] KTOP

KTOP is integer scalar

[in] KBOT

KBOT is integer scalar
These are the first and last rows and columns of an
isolated diagonal block upon which the QR sweep is to be
applied. It is assumed without a check that
either KTOP = 1 or $H(KTOP, KTOP-1) = 0$
and
either KBOT = N or $H(KBOT+1, KBOT) = 0$.

[in] NSHFTS

NSHFTS is integer scalar

NSHFTS gives the number of simultaneous shifts. NSHFTS must be positive and even.

[in,out] S

S is COMPLEX*16 array of size (NSHFTS)
S contains the shifts of origin that define the multi-shift QR sweep. On output S may be reordered.

[in,out] H

H is COMPLEX*16 array of size (LDH,N)
On input H contains a Hessenberg matrix. On output a multi-shift QR sweep with shifts $SR(J)+i*SI(J)$ is applied to the isolated diagonal block in rows and columns KTOP through KBOT.

[in] LDH

LDH is integer scalar
LDH is the leading dimension of H just as declared in the calling procedure. $LDH \geq \max(1, N)$.

[in] ILOZ

ILOZ is INTEGER

[in] IHIZ

IHIZ is INTEGER
Specify the rows of Z to which transformations must be applied if WANTZ is .TRUE.. $1 \leq ILOZ \leq IHIZ \leq N$

[in,out] Z

Z is COMPLEX*16 array of size (LDZ,IHI)
If WANTZ = .TRUE., then the QR Sweep unitary similarity transformation is accumulated into $Z(ILOZ:IHIZ, ILO:IHI)$ from the right.
If WANTZ = .FALSE., then Z is unreferenced.

[in] LDZ

LDZ is integer scalar
LDA is the leading dimension of Z just as declared in the calling procedure. $LDZ \geq N$.

[out] V

V is COMPLEX*16 array of size (LDV,NSHFTS/2)

[in] LDV

LDV is integer scalar
LDV is the leading dimension of V as declared in the
calling procedure. LDV.GE.3.

[out] U

U is COMPLEX*16 array of size
(LDU,3*NSHFTS-3)

[in] LDU

LDU is integer scalar
LDU is the leading dimension of U just as declared in the
in the calling subroutine. LDU.GE.3*NSHFTS-3.

[in] NH

NH is integer scalar
NH is the number of columns in array WH available for
workspace. NH.GE.1.

[out] WH

WH is COMPLEX*16 array of size (LDWH,NH)

[in] LDWH

LDWH is integer scalar
Leading dimension of WH just as declared in the
calling procedure. LDWH.GE.3*NSHFTS-3.

[in] NV

NV is integer scalar
NV is the number of rows in WV available for workspace.
NV.GE.1.

[out] WV

WV is COMPLEX*16 array of size
(LDWV,3*NSHFTS-3)

[in] LDWV

LDWV is integer scalar
LDWV is the leading dimension of WV as declared in the
in the calling subroutine. LDWV.GE.NV.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Contributors:

=====

Karen Braman and Ralph Byers, Department of Mathematics,
 University of Kansas, USA

References:

=====

K. Braman, R. Byers and R. Mathias, The Multi-Shift QR
 Algorithm Part I: Maintaining Well Focused Shifts, and Level 3
 Performance, SIAM Journal of Matrix Analysis, volume 23, pages
 929--947, 2002.

— zlaqr5.f —

```
* =====
*      SUBROUTINE ZLAQR5( WANTT, WANTZ, KACC22, N, KTOP, KBOT, NSHFTS, S,
*      $                  H, LDH, ILOZ, IHIZ, Z, LDZ, V, LDV, U, LDU, NV,
*      $                  WV, LDWV, NH, WH, LDWH )
*
*  -- LAPACK auxiliary routine (version 3.4.0) --
*  -- LAPACK is a software package provided by Univ. of Tennessee,    --
*  -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*  November 2011
*
*  .. Scalar Arguments ..
*  INTEGER          IHIZ, ILOZ, KACC22, KBOT, KTOP, LDH, LDU, LDV,
*  $                LDWH, LDWV, LDZ, N, NH, NSHFTS, NV
*  LOGICAL          WANTT, WANTZ
*
*  ..
*  .. Array Arguments ..
*  COMPLEX*16       H( LDH, * ), S( * ), U( LDU, * ), V( LDV, * ),
*  $                WH( LDWH, * ), WV( LDWV, * ), Z( LDZ, * )
*
*  ..
```

```

*
* =====
* .. Parameters ..
*   COMPLEX*16      ZERO, ONE
*   PARAMETER      ( ZERO = ( 0.0d0, 0.0d0 ),
* $               ONE = ( 1.0d0, 0.0d0 ) )
*   DOUBLE PRECISION RZERO, RONE
*   PARAMETER      ( RZERO = 0.0d0, RONE = 1.0d0 )
*
* ..
* .. Local Scalars ..
*   COMPLEX*16      ALPHA, BETA, CDUM, REFSUM
*   DOUBLE PRECISION H11, H12, H21, H22, SAFMAX, SAFMIN, SCL,
* $               SMLNUM, TST1, TST2, ULP
*   INTEGER         I2, I4, INCOL, J, J2, J4, JBOT, JCOL, JLEN,
* $               JROW, JTOP, K, K1, KDU, KMS, KNZ, KRCOL, KZS,
* $               M, M22, MBOT, MEND, MSTART, MTOP, NBMPS, NDCOL,
* $               NS, NU
*   LOGICAL         ACCUM, BLK22, BMP22
*
* ..
* .. External Functions ..
*   DOUBLE PRECISION DLAMCH
*   EXTERNAL        DLAMCH
*
* ..
* .. Intrinsic Functions ..
*
*   INTRINSIC        ABS, DBLE, DCONJG, DIMAG, MAX, MIN, MOD
*
* ..
* .. Local Arrays ..
*   COMPLEX*16      VT( 3 )
*
* ..
* .. External Subroutines ..
*   EXTERNAL        DLABAD, ZGEMM, ZLACPY, ZLAQR1, ZLARFG, ZLASET,
* $               ZTRMM
*
* ..
* .. Statement Functions ..
*   DOUBLE PRECISION CABS1
*
* ..
* .. Statement Function definitions ..
*   CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
* ..
* .. Executable Statements ..
*
*   ==== If there are no shifts, then there is nothing to do. ====
*
*   IF( NSHFTS.LT.2 )
* $     RETURN
*
*
*   ==== If the active block is empty or 1-by-1, then there
*   .   is nothing to do. ====
*

```

```

      IF( KTOP.GE.KBOT )
$     RETURN
*
*     ==== NSHFTS is supposed to be even, but if it is odd,
*     .   then simply reduce it by one.  ====
*
      NS = NSHFTS - MOD( NSHFTS, 2 )
*
*     ==== Machine constants for deflation ====
*
      SAFMIN = DLAMCH( 'SAFE MINIMUM' )
      SAFMAX = RONE / SAFMIN
      CALL DLABAD( SAFMIN, SAFMAX )
      ULP = DLAMCH( 'PRECISION' )
      SMLNUM = SAFMIN*( DBLE( N ) / ULP )
*
*     ==== Use accumulated reflections to update far-from-diagonal
*     .   entries ? ====
*
      ACCUM = ( KACC22.EQ.1 ) .OR. ( KACC22.EQ.2 )
*
*     ==== If so, exploit the 2-by-2 block structure? ====
*
      BLK22 = ( NS.GT.2 ) .AND. ( KACC22.EQ.2 )
*
*     ==== clear trash ====
*
      IF( KTOP+2.LE.KBOT )
$     H( KTOP+2, KTOP ) = ZERO
*
*     ==== NBMP5 = number of 2-shift bulges in the chain ====
*
      NBMP5 = NS / 2
*
*     ==== KDU = width of slab ====
*
      KDU = 6*NBMPS - 3
*
*     ==== Create and chase chains of NBMP5 bulges ====
*
      DO 210 INCOL = 3*( 1-NBMPS ) + KTOP - 1, KBOT - 2, 3*NBMPS - 2
        NDCOL = INCOL + KDU
        IF( ACCUM )
$          CALL ZLASET( 'ALL', KDU, KDU, ZERO, ONE, U, LDU )
*
*     ==== Near-the-diagonal bulge chase.  The following loop
*     .   performs the near-the-diagonal part of a small bulge
*     .   multi-shift QR sweep.  Each 6*NBMPS-2 column diagonal
*     .   chunk extends from column INCOL to column NDCOL
*     .   (including both column INCOL and column NDCOL).  The

```

```

*      .   following loop chases a 3*NBMPS column long chain of
*      .   NBMPS bulges 3*NBMPS-2 columns to the right. (INCOL
*      .   may be less than KTOP and NDCOL may be greater than
*      .   KBOT indicating phantom columns from which to chase
*      .   bulges before they are actually introduced or to which
*      .   to chase bulges beyond column KBOT.)  ====
*
DO 140 KRCOL = INCOL, MIN( INCOL+3*NBMPS-3, KBOT-2 )

*
*      ==== Bulges number MTOP to MBOT are active double implicit
*      .   shift bulges. There may or may not also be small
*      .   2-by-2 bulge, if there is room. The inactive bulges
*      .   (if any) must wait until the active bulges have moved
*      .   down the diagonal to make room. The phantom matrix
*      .   paradigm described above helps keep track.  ====
*
      MTOP = MAX( 1, ( ( KTOP-1 )-KRCOL+2 ) / 3+1 )
      MBOT = MIN( NBMPS, ( KBOT-KRCOL ) / 3 )
      M22 = MBOT + 1
      BMP22 = ( MBOT.LT.NBMPS ) .AND. ( KRCOL+3*( M22-1 ) ).EQ.
$          ( KBOT-2 )

*
*      ==== Generate reflections to chase the chain right
*      .   one column. (The minimum value of K is KTOP-1.)  ====
*
DO 10 M = MTOP, MBOT
      K = KRCOL + 3*( M-1 )
      IF( K.EQ.KTOP-1 ) THEN
          CALL ZLAQR1( 3, H( KTOP, KTOP ), LDH, S( 2*M-1 ),
$              S( 2*M ), V( 1, M ) )
          ALPHA = V( 1, M )
          CALL ZLARFG( 3, ALPHA, V( 2, M ), 1, V( 1, M ) )
      ELSE
          BETA = H( K+1, K )
          V( 2, M ) = H( K+2, K )
          V( 3, M ) = H( K+3, K )
          CALL ZLARFG( 3, BETA, V( 2, M ), 1, V( 1, M ) )

*
*      ==== A Bulge may collapse because of vigilant
*      .   deflation or destructive underflow. In the
*      .   underflow case, try the two-small-subdiagonals
*      .   trick to try to reinflate the bulge.  ====
*
      IF( H( K+3, K ).NE.ZERO .OR. H( K+3, K+1 ).NE.
$          ZERO .OR. H( K+3, K+2 ).EQ.ZERO ) THEN

*
*      ==== Typical case: not collapsed (yet).  ====
*
          H( K+1, K ) = BETA
          H( K+2, K ) = ZERO

```



```

      H( K+3, K ) = ZERO
    ELSE
*
*      ==== Atypical case: collapsed. Attempt to
*      . reintroduce ignoring H(K+1,K) and H(K+2,K).
*      . If the fill resulting from the new
*      . reflector is too large, then abandon it.
*      . Otherwise, use the new one. ====
*
      CALL ZLAQR1( 3, H( K+1, K+1 ), LDH, S( 2*M-1 ),
$              S( 2*M ), VT )
      ALPHA = VT( 1 )
      CALL ZLARFG( 3, ALPHA, VT( 2 ), 1, VT( 1 ) )
      REFSUM = DCONJG( VT( 1 ) )*
$              ( H( K+1, K )+DCONJG( VT( 2 ) )*)
$              H( K+2, K ) )
*
      IF( CABS1( H( K+2, K )-REFSUM*VT( 2 ) )+
$          CABS1( REFSUM*VT( 3 ) ).GT.ULP*
$          ( CABS1( H( K, K ) )+CABS1( H( K+1,
$          K+1 ) )+CABS1( H( K+2, K+2 ) ) ) ) THEN
*
*      ==== Starting a new bulge here would
*      . create non-negligible fill. Use
*      . the old one with trepidation. ====
*
      H( K+1, K ) = BETA
      H( K+2, K ) = ZERO
      H( K+3, K ) = ZERO
    ELSE
*
*      ==== Stating a new bulge here would
*      . create only negligible fill.
*      . Replace the old reflector with
*      . the new one. ====
*
      H( K+1, K ) = H( K+1, K ) - REFSUM
      H( K+2, K ) = ZERO
      H( K+3, K ) = ZERO
      V( 1, M ) = VT( 1 )
      V( 2, M ) = VT( 2 )
      V( 3, M ) = VT( 3 )
      END IF
      END IF
      END IF
10    CONTINUE
*
*      ==== Generate a 2-by-2 reflection, if needed. ====
*
      K = KRCOL + 3*( M22-1 )

```

```

IF( BMP22 ) THEN
  IF( K.EQ.KTOP-1 ) THEN
    CALL ZLAQR1( 2, H( K+1, K+1 ), LDH, S( 2*M22-1 ),
$           S( 2*M22 ), V( 1, M22 ) )
    BETA = V( 1, M22 )
    CALL ZLARFG( 2, BETA, V( 2, M22 ), 1, V( 1, M22 ) )
  ELSE
    BETA = H( K+1, K )
    V( 2, M22 ) = H( K+2, K )
    CALL ZLARFG( 2, BETA, V( 2, M22 ), 1, V( 1, M22 ) )
    H( K+1, K ) = BETA
    H( K+2, K ) = ZERO
  END IF
END IF

*
*      ==== Multiply H by reflections from the left ====
*

IF( ACCUM ) THEN
  JBOT = MIN( NDCOL, KBOT )
ELSE IF( WANTT ) THEN
  JBOT = N
ELSE
  JBOT = KBOT
END IF
DO 30 J = MAX( KTOP, KRCOL ), JBOT
  MEND = MIN( MBOT, ( J-KRCOL+2 ) / 3 )
  DO 20 M = MTOP, MEND
    K = KRCOL + 3*( M-1 )
    REFSUM = DCONJG( V( 1, M ) )*
$           ( H( K+1, J )+DCONJG( V( 2, M ) )*)
$           H( K+2, J )+DCONJG( V( 3, M ) )*H( K+3, J ) )
    H( K+1, J ) = H( K+1, J ) - REFSUM
    H( K+2, J ) = H( K+2, J ) - REFSUM*V( 2, M )
    H( K+3, J ) = H( K+3, J ) - REFSUM*V( 3, M )
20  CONTINUE
30  CONTINUE
IF( BMP22 ) THEN
  K = KRCOL + 3*( M22-1 )
  DO 40 J = MAX( K+1, KTOP ), JBOT
    REFSUM = DCONJG( V( 1, M22 ) )*
$           ( H( K+1, J )+DCONJG( V( 2, M22 ) )*)
$           H( K+2, J ) )
    H( K+1, J ) = H( K+1, J ) - REFSUM
    H( K+2, J ) = H( K+2, J ) - REFSUM*V( 2, M22 )
40  CONTINUE
END IF

*
*      ==== Multiply H by reflections from the right.
*      .      Delay filling in the last row until the
*      .      vigilant deflation check is complete. ====

```

```

*
      IF( ACCUM ) THEN
        JTOP = MAX( KTOP, INCOL )
      ELSE IF( WANTT ) THEN
        JTOP = 1
      ELSE
        JTOP = KTOP
      END IF
      DO 80 M = MTOP, MBOT
        IF( V( 1, M ).NE.ZERO ) THEN
          K = KRCOL + 3*( M-1 )
          DO 50 J = JTOP, MIN( KBOT, K+3 )
            REFSUM = V( 1, M )*( H( J, K+1 )+V( 2, M )*
$              H( J, K+2 )+V( 3, M )*H( J, K+3 ) )
            H( J, K+1 ) = H( J, K+1 ) - REFSUM
            H( J, K+2 ) = H( J, K+2 ) -
$              REFSUM*DCONJG( V( 2, M ) )
            H( J, K+3 ) = H( J, K+3 ) -
$              REFSUM*DCONJG( V( 3, M ) )
50          CONTINUE
        *
        IF( ACCUM ) THEN
          *
          *      ==== Accumulate U. (If necessary, update Z later
          *      .      with with an efficient matrix-matrix
          *      .      multiply.) ====
          *
          KMS = K - INCOL
          DO 60 J = MAX( 1, KTOP-INCOL ), KDU
            REFSUM = V( 1, M )*( U( J, KMS+1 )+V( 2, M )*
$              U( J, KMS+2 )+V( 3, M )*U( J, KMS+3 ) )
            U( J, KMS+1 ) = U( J, KMS+1 ) - REFSUM
            U( J, KMS+2 ) = U( J, KMS+2 ) -
$              REFSUM*DCONJG( V( 2, M ) )
            U( J, KMS+3 ) = U( J, KMS+3 ) -
$              REFSUM*DCONJG( V( 3, M ) )
60          CONTINUE
        ELSE IF( WANTZ ) THEN
          *
          *      ==== U is not accumulated, so update Z
          *      .      now by multiplying by reflections
          *      .      from the right. ====
          *
          DO 70 J = ILOZ, IHIZ
            REFSUM = V( 1, M )*( Z( J, K+1 )+V( 2, M )*
$              Z( J, K+2 )+V( 3, M )*Z( J, K+3 ) )
            Z( J, K+1 ) = Z( J, K+1 ) - REFSUM
            Z( J, K+2 ) = Z( J, K+2 ) -
$              REFSUM*DCONJG( V( 2, M ) )
            Z( J, K+3 ) = Z( J, K+3 ) -

```

```

$                                REFSUM*DCONJG( V( 3, M ) )
70          CONTINUE
          END IF
          END IF
80      CONTINUE
*
*      ==== Special case: 2-by-2 reflection (if needed) ====
*
      K = KRCOL + 3*( M22-1 )
      IF( BMP22 ) THEN
          IF ( V( 1, M22 ).NE.ZERO ) THEN
              DO 90 J = JTOP, MIN( KBOT, K+3 )
                  REFSUM = V( 1, M22 )*( H( J, K+1 )+V( 2, M22 )*
$                      H( J, K+2 ) )
                  H( J, K+1 ) = H( J, K+1 ) - REFSUM
                  H( J, K+2 ) = H( J, K+2 ) -
$                      REFSUM*DCONJG( V( 2, M22 ) )
90          CONTINUE
*
          IF( ACCUM ) THEN
              KMS = K - INCOL
              DO 100 J = MAX( 1, KTOP-INCOL ), KDU
                  REFSUM = V( 1, M22 )*( U( J, KMS+1 )+
$                      V( 2, M22 )*U( J, KMS+2 ) )
                  U( J, KMS+1 ) = U( J, KMS+1 ) - REFSUM
                  U( J, KMS+2 ) = U( J, KMS+2 ) -
$                      REFSUM*DCONJG( V( 2, M22 ) )
100         CONTINUE
          ELSE IF( WANTZ ) THEN
              DO 110 J = ILOZ, IHIZ
                  REFSUM = V( 1, M22 )*( Z( J, K+1 )+V( 2, M22 )*
$                      Z( J, K+2 ) )
                  Z( J, K+1 ) = Z( J, K+1 ) - REFSUM
                  Z( J, K+2 ) = Z( J, K+2 ) -
$                      REFSUM*DCONJG( V( 2, M22 ) )
110         CONTINUE
          END IF
          END IF
          END IF
*
*      ==== Vigilant deflation check ====
*
      MSTART = MTOP
      IF( KRCOL+3*( MSTART-1 ).LT.KTOP )
$          MSTART = MSTART + 1
      MEND = MBOT
      IF( BMP22 )
$          MEND = MEND + 1
      IF( KRCOL.EQ.KBOT-2 )
$          MEND = MEND + 1

```

```

DO 120 M = MSTART, MEND
    K = MIN( KBOT-1, KRCOL+3*( M-1 ) )

*
*      ==== The following convergence test requires that
*      . the tradition small-compared-to-nearby-diagonals
*      . criterion and the Ahues & Tisseur (LAWN 122, 1997)
*      . criteria both be satisfied. The latter improves
*      . accuracy in some examples. Falling back on an
*      . alternate convergence criterion when TST1 or TST2
*      . is zero (as done here) is traditional but probably
*      . unnecessary. ====
*

    IF( H( K+1, K ).NE.ZERO ) THEN
        TST1 = CABS1( H( K, K ) ) + CABS1( H( K+1, K+1 ) )
        IF( TST1.EQ.RZERO ) THEN
            IF( K.GE.KTOP+1 )
                $      TST1 = TST1 + CABS1( H( K, K-1 ) )
            IF( K.GE.KTOP+2 )
                $      TST1 = TST1 + CABS1( H( K, K-2 ) )
            IF( K.GE.KTOP+3 )
                $      TST1 = TST1 + CABS1( H( K, K-3 ) )
            IF( K.LE.KBOT-2 )
                $      TST1 = TST1 + CABS1( H( K+2, K+1 ) )
            IF( K.LE.KBOT-3 )
                $      TST1 = TST1 + CABS1( H( K+3, K+1 ) )
            IF( K.LE.KBOT-4 )
                $      TST1 = TST1 + CABS1( H( K+4, K+1 ) )
            END IF
            IF( CABS1( H( K+1, K ) ).LE.MAX( SMLNUM, ULP*TST1 ) )
                $      THEN
                    H12 = MAX( CABS1( H( K+1, K ) ),
                $      CABS1( H( K, K+1 ) ) )
                    H21 = MIN( CABS1( H( K+1, K ) ),
                $      CABS1( H( K, K+1 ) ) )
                    H11 = MAX( CABS1( H( K+1, K+1 ) ),
                $      CABS1( H( K, K )-H( K+1, K+1 ) ) )
                    H22 = MIN( CABS1( H( K+1, K+1 ) ),
                $      CABS1( H( K, K )-H( K+1, K+1 ) ) )
                    SCL = H11 + H12
                    TST2 = H22*( H11 / SCL )

*
                IF( TST2.EQ.RZERO .OR. H21*( H12 / SCL ).LE.
                $      MAX( SMLNUM, ULP*TST2 ) )H( K+1, K ) = ZERO

                    END IF
                END IF
            CONTINUE
120
*
*      ==== Fill in the last row of each bulge. ====
*

MEND = MIN( NBMP5, ( KBOT-KRCOL-1 ) / 3 )

```

```

DO 130 M = MTOP, MEND
    K = KRCOL + 3*( M-1 )
    REFSUM = V( 1, M )*V( 3, M )*H( K+4, K+3 )
    H( K+4, K+1 ) = -REFSUM
    H( K+4, K+2 ) = -REFSUM*DCONJG( V( 2, M ) )
    H( K+4, K+3 ) = H( K+4, K+3 ) -
$           REFSUM*DCONJG( V( 3, M ) )
130    CONTINUE
*
*       ==== End of near-the-diagonal bulge chase. ====
*
140    CONTINUE
*
*       ==== Use U (if accumulated) to update far-from-diagonal
*       .     entries in H.  If required, use U to update Z as
*       .     well. ====
*
IF( ACCUM ) THEN
    IF( WANTT ) THEN
        JTOP = 1
        JBOT = N
    ELSE
        JTOP = KTOP
        JBOT = KBOT
    END IF
    IF( ( .NOT.BLK22 ) .OR. ( INCOL.LT.KTOP ) .OR.
$       ( NDCOL.GT.KBOT ) .OR. ( NS.LE.2 ) ) THEN
*
*       ==== Updates not exploiting the 2-by-2 block
*       .     structure of U.  K1 and NU keep track of
*       .     the location and size of U in the special
*       .     cases of introducing bulges and chasing
*       .     bulges off the bottom.  In these special
*       .     cases and in case the number of shifts
*       .     is NS = 2, there is no 2-by-2 block
*       .     structure to exploit. ====
*
        K1 = MAX( 1, KTOP-INCOL )
        NU = ( KDU-MAX( 0, NDCOL-KBOT ) ) - K1 + 1
*
*       ==== Horizontal Multiply ====
*
        DO 150 JCOL = MIN( NDCOL, KBOT ) + 1, JBOT, NH
            JLEN = MIN( NH, JBOT-JCOL+1 )
            CALL ZGEMM( 'C', 'N', NU, JLEN, NU, ONE, U( K1, K1 ),
$                   LDU, H( INCOL+K1, JCOL ), LDH, ZERO, WH,
$                   LDWH )
            CALL ZLACPY( 'ALL', NU, JLEN, WH, LDWH,
$                   H( INCOL+K1, JCOL ), LDH )
150        CONTINUE

```

```

*
*      ==== Vertical multiply ====
*
      DO 160 JROW = JTOP, MAX( KTOP, INCOL ) - 1, NV
          JLEN = MIN( NV, MAX( KTOP, INCOL )-JROW )
          CALL ZGEMM( 'N', 'N', JLEN, NU, NU, ONE,
$              H( JROW, INCOL+K1 ), LDH, U( K1, K1 ),
$              LDU, ZERO, WV, LDWV )
          CALL ZLACPY( 'ALL', JLEN, NU, WV, LDWV,
$              H( JROW, INCOL+K1 ), LDH )
160      CONTINUE
*
*      ==== Z multiply (also vertical) ====
*
      IF( WANTZ ) THEN
          DO 170 JROW = ILOZ, IHIZ, NV
              JLEN = MIN( NV, IHIZ-JROW+1 )
              CALL ZGEMM( 'N', 'N', JLEN, NU, NU, ONE,
$                  Z( JROW, INCOL+K1 ), LDZ, U( K1, K1 ),
$                  LDU, ZERO, WV, LDWV )
              CALL ZLACPY( 'ALL', JLEN, NU, WV, LDWV,
$                  Z( JROW, INCOL+K1 ), LDZ )
170          CONTINUE
          END IF
      ELSE
*
*      ==== Updates exploiting U's 2-by-2 block structure.
*      .   (I2, I4, J2, J4 are the last rows and columns
*      .   of the blocks.) ====
*
          I2 = ( KDU+1 ) / 2
          I4 = KDU
          J2 = I4 - I2
          J4 = KDU
*
*      ==== KZS and KNZ deal with the band of zeros
*      .   along the diagonal of one of the triangular
*      .   blocks. ====
*
          KZS = ( J4-J2 ) - ( NS+1 )
          KNZ = NS + 1
*
*      ==== Horizontal multiply ====
*
      DO 180 JCOL = MIN( NDCOL, KBOT ) + 1, JBOT, NH
          JLEN = MIN( NH, JBOT-JCOL+1 )
*
*      ==== Copy bottom of H to top+KZS of scratch ====
*      (The first KZS rows get multiplied by zero.) ====
*

```

```

$          CALL ZLACPY( 'ALL', KNZ, JLEN, H( INCOL+1+J2, JCOL ),
$              LDH, WH( KZS+1, 1 ), LDWH )
*
*          ===== Multiply by U21**H =====
*
$          CALL ZLASET( 'ALL', KZS, JLEN, ZERO, ZERO, WH, LDWH )
$          CALL ZTRMM( 'L', 'U', 'C', 'N', KNZ, JLEN, ONE,
$              U( J2+1, 1+KZS ), LDU, WH( KZS+1, 1 ),
$              LDWH )
*
*          ===== Multiply top of H by U11**H =====
*
$          CALL ZGEMM( 'C', 'N', I2, JLEN, J2, ONE, U, LDU,
$              H( INCOL+1, JCOL ), LDH, ONE, WH, LDWH )
*
*          ===== Copy top of H to bottom of WH =====
*
$          CALL ZLACPY( 'ALL', J2, JLEN, H( INCOL+1, JCOL ), LDH,
$              WH( I2+1, 1 ), LDWH )
*
*          ===== Multiply by U21**H =====
*
$          CALL ZTRMM( 'L', 'L', 'C', 'N', J2, JLEN, ONE,
$              U( 1, I2+1 ), LDU, WH( I2+1, 1 ), LDWH )
*
*          ===== Multiply by U22 =====
*
$          CALL ZGEMM( 'C', 'N', I4-I2, JLEN, J4-J2, ONE,
$              U( J2+1, I2+1 ), LDU,
$              H( INCOL+1+J2, JCOL ), LDH, ONE,
$              WH( I2+1, 1 ), LDWH )
*
*          ===== Copy it back =====
*
$          CALL ZLACPY( 'ALL', KDU, JLEN, WH, LDWH,
$              H( INCOL+1, JCOL ), LDH )
180      CONTINUE
*
*          ===== Vertical multiply =====
*
$          DO 190 JROW = JTOP, MAX( INCOL, KTOP ) - 1, NV
$              JLEN = MIN( NV, MAX( INCOL, KTOP )-JROW )
*
*          ===== Copy right of H to scratch (the first KZS
*              .   columns get multiplied by zero) =====
*
$          CALL ZLACPY( 'ALL', JLEN, KNZ, H( JROW, INCOL+1+J2 ),
$              LDH, WV( 1, 1+KZS ), LDWV )
*
*          ===== Multiply by U21 =====

```



```

*
*      CALL ZLASET( 'ALL', JLEN, KZS, ZERO, ZERO, WV, LDWV )
*      CALL ZTRMM( 'R', 'U', 'N', 'N', JLEN, KNZ, ONE,
$           U( J2+1, 1+KZS ), LDU, WV( 1, 1+KZS ),
$           LDWV )
*
*      ==== Multiply by U11 ====
*
*      CALL ZGEMM( 'N', 'N', JLEN, I2, J2, ONE,
$           H( JROW, INCOL+1 ), LDH, U, LDU, ONE, WV,
$           LDWV )
*
*      ==== Copy left of H to right of scratch ====
*
*      CALL ZLACPY( 'ALL', JLEN, J2, H( JROW, INCOL+1 ), LDH,
$           WV( 1, 1+I2 ), LDWV )
*
*      ==== Multiply by U21 ====
*
*      CALL ZTRMM( 'R', 'L', 'N', 'N', JLEN, I4-I2, ONE,
$           U( 1, I2+1 ), LDU, WV( 1, 1+I2 ), LDWV )
*
*      ==== Multiply by U22 ====
*
*      CALL ZGEMM( 'N', 'N', JLEN, I4-I2, J4-J2, ONE,
$           H( JROW, INCOL+1+J2 ), LDH,
$           U( J2+1, I2+1 ), LDU, ONE, WV( 1, 1+I2 ),
$           LDWV )
*
*      ==== Copy it back ====
*
*      CALL ZLACPY( 'ALL', JLEN, KDU, WV, LDWV,
$           H( JROW, INCOL+1 ), LDH )
190      CONTINUE
*
*      ==== Multiply Z (also vertical) ====
*
*      IF( WANTZ ) THEN
*      DO 200 JROW = ILOZ, IHIZ, NV
*          JLEN = MIN( NV, IHIZ-JROW+1 )
*
*      ==== Copy right of Z to left of scratch (first
*      .      KZS columns get multiplied by zero) ====
*
*      CALL ZLACPY( 'ALL', JLEN, KNZ,
$           Z( JROW, INCOL+1+J2 ), LDZ,
$           WV( 1, 1+KZS ), LDWV )
*
*      ==== Multiply by U12 ====
*

```

```

                CALL ZLASET( 'ALL', JLEN, KZS, ZERO, ZERO, WV,
$                  LDWV )
                CALL ZTRMM( 'R', 'U', 'N', 'N', JLEN, KNZ, ONE,
$                  U( J2+1, 1+KZS ), LDU, WV( 1, 1+KZS ),
$                  LDWV )
*
*          ==== Multiply by U11 ====
*
                CALL ZGEMM( 'N', 'N', JLEN, I2, J2, ONE,
$                  Z( JROW, INCOL+1 ), LDZ, U, LDU, ONE,
$                  WV, LDWV )
*
*          ==== Copy left of Z to right of scratch ====
*
                CALL ZLACPY( 'ALL', JLEN, J2, Z( JROW, INCOL+1 ),
$                  LDZ, WV( 1, 1+I2 ), LDWV )
*
*          ==== Multiply by U21 ====
*
                CALL ZTRMM( 'R', 'L', 'N', 'N', JLEN, I4-I2, ONE,
$                  U( 1, I2+1 ), LDU, WV( 1, 1+I2 ),
$                  LDWV )
*
*          ==== Multiply by U22 ====
*
                CALL ZGEMM( 'N', 'N', JLEN, I4-I2, J4-J2, ONE,
$                  Z( JROW, INCOL+1+J2 ), LDZ,
$                  U( J2+1, I2+1 ), LDU, ONE,
$                  WV( 1, 1+I2 ), LDWV )
*
*          ==== Copy the result back to Z ====
*
                CALL ZLACPY( 'ALL', JLEN, KDU, WV, LDWV,
$                  Z( JROW, INCOL+1 ), LDZ )
200          CONTINUE
              END IF
            END IF
          END IF
210 CONTINUE
*
*          ==== End of ZLAQR5 ====
*
        END

```

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)) (rzero 0.0d0)
   (rone 1.0d0))
  (declare (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
           (type (double-float 0.0d0 0.0d0) rzero)
           (type (double-float 1.0d0 1.0d0) rone) (ignorable zero one rzero rone))
  (defun zlaqr5
    (wantt wantz kacc22 n ktop kbot nshfts s h ldh iloz ihiz z ldz v ldv u ldu nv
     wv ldwv nh wh ldwh)
    (declare (type f2cl-lib:logical wantz wantt)
             (type (f2cl-lib:integer4) ldwh nh ldwv nv ldu ldv ldz ihiz iloz ldh nshfts
                  kbot ktop n kacc22)
             (type (array f2cl-lib:complex16 (*)) wh wv u v z h s))
    (f2cl-lib:with-multi-array-data
      ((s f2cl-lib:complex16 s-%data% s-%offset%)
       (h f2cl-lib:complex16 h-%data% h-%offset%)
       (z f2cl-lib:complex16 z-%data% z-%offset%)
       (v f2cl-lib:complex16 v-%data% v-%offset%)
       (u f2cl-lib:complex16 u-%data% u-%offset%)
       (wv f2cl-lib:complex16 wv-%data% wv-%offset%)
       (wh f2cl-lib:complex16 wh-%data% wh-%offset%))
      (labels
        ((cabs1 (cdum) (+ (abs (f2cl-lib:db1e cdum))
                          (abs (f2cl-lib:dimag cdum)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                           (values double-float &rest t)) cabs1))
        (prog
          ((vt (make-array 3 :element-type 'f2cl-lib:complex16)) (accum nil)
           (blk22 nil) (bmp22 nil) (i2 0) (i4 0) (incol 0) (j 0) (j2 0) (j4 0)
           (jbot 0)
           (jcol 0) (jlen 0) (jrow 0) (jtop 0) (k 0) (k1 0) (kdu 0) (kms 0)
           (knz 0)
           (krcol 0) (kzs 0) (m 0) (m22 0) (mbot 0) (mend 0) (mstart 0) (mtop 0)
           (nbmps 0) (ndcol 0) (ns 0) (nu 0) (h11 0.0d0) (h12 0.0d0) (h21 0.0d0)
           (h22 0.0d0) (safmax 0.0d0) (safmin 0.0d0) (scl 0.0d0) (smlnum 0.0d0)
           (tst1 0.0d0) (tst2 0.0d0) (ulp 0.0d0) (alpha #C(0.0d0 0.0d0))
           (beta #C(0.0d0 0.0d0)) (cdum #C(0.0d0 0.0d0))
           (refsum #C(0.0d0 0.0d0)))
          (declare (type (array f2cl-lib:complex16 (3)) vt)
                   (type f2cl-lib:logical bmp22 blk22 accum)
                   (type (f2cl-lib:integer4) nu ns ndcol nbmps mtop mstart
                        mend mbot m22 m kzs
                        krcol knz kms kdu k1 k jtop jrow jlen jcol jbot j4 j2 j incol i4 i2)
                   (type (double-float) ulp tst2 tst1 smlnum scl safmin safmax
                        h22 h21 h12 h11)
                   (type (f2cl-lib:complex16) refsum cdum beta alpha))
          (if (< nshfts 2) (go end_label)) (if (>= ktop kbot) (go end_label))
          (setf ns (f2cl-lib:int-sub nshfts (mod nshfts 2))))

```

```

(setf safmin (dlamch "SAFE MINIMUM")) (setf safmax (/ rone safmin))
(multiple-value-bind (var-0 var-1)
  (dlabad safmin safmax) (declare (ignore))
  (when var-0 (setf safmin var-0)) (when var-1 (setf safmax var-1)))
(setf ulp (dlamch "PRECISION"))
(setf smlnum (* safmin (/ (f2cl-lib:double n) ulp)))
(setf accum (or (= kacc22 1) (= kacc22 2)))
(setf blk22 (and (> ns 2) (= kacc22 2)))
(if (<= (f2cl-lib:int-add ktop 2) kbot)
  (setf
    (f2cl-lib:fref h-%data% ((f2cl-lib:int-add ktop 2) ktop)
      ((1 ldh) (1 *))
      h-%offset%)
    zero))
(setf nbmps (the f2cl-lib:integer4 (truncate ns 2)))
(setf kdu (f2cl-lib:int-sub (f2cl-lib:int-mul 6 nbmps) 3))
(f2cl-lib:fdo (incol
  (f2cl-lib:int-add
    (f2cl-lib:int-mul 3 (f2cl-lib:int-add 1
      (f2cl-lib:int-sub nbmps))) ktop
    (f2cl-lib:int-sub 1))
  (f2cl-lib:int-add incol
    (f2cl-lib:int-add (f2cl-lib:int-mul 3 nbmps)
      (f2cl-lib:int-sub 2))))
  ((>
    incol (f2cl-lib:int-add kbot (f2cl-lib:int-sub 2)))
    nil)
  (tagbody (setf ndcol (f2cl-lib:int-add incol kdu))
    (if accum
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6)
        (zlaset "ALL" kdu kdu zero one u ldu)
        (declare (ignore var-0 var-5))
        (when var-1 (setf kdu var-1))
        (when var-2 (setf kdu var-2))
        (when var-3 (setf zero var-3))
        (when var-4 (setf one var-4))
        (when var-6 (setf ldu var-6))))
      (f2cl-lib:fdo (krcol incol (f2cl-lib:int-add krcol 1))
        ((> krcol
          (min
            (the f2cl-lib:integer4
              (f2cl-lib:int-add incol (f2cl-lib:int-mul 3 nbmps)
                (f2cl-lib:int-sub 3)))
            (the f2cl-lib:integer4
              (f2cl-lib:int-add kbot (f2cl-lib:int-sub 2))))))
          nil)
        (tagbody
          (setf mtop
            (max 1 (+ (the f2cl-lib:integer4

```

```

                (truncate (+ (- ktop 1 krcol) 2) 3)) 1)))
(setf mbot (min nbmps (the f2cl-lib:integer4
                        (truncate (- kbot krcol) 3))))
(setf m22 (f2cl-lib:int-add mbot 1))
(setf bmp22
  (and (< mbot nbmps)
    (= (f2cl-lib:int-add krcol
      (f2cl-lib:int-mul 3 (f2cl-lib:int-sub m22 1)))
      (f2cl-lib:int-sub kbot 2))))
(f2cl-lib:fdo (m mtop (f2cl-lib:int-add m 1))
  (> m mbot) nil)
(tagbody
  (setf k
    (f2cl-lib:int-add krcol
      (f2cl-lib:int-mul 3 (f2cl-lib:int-sub m 1))))
  (cond
    ((= k (f2cl-lib:int-add ktop (f2cl-lib:int-sub 1)))
      (zlaqr1 3
        (f2cl-lib:array-slice h-%data%
          f2cl-lib:complex16 (ktop ktop)
          ((1 ldh) (1 *)) h-%offset%)
        ldh
        (f2cl-lib:fref s-%data%
          ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 m) 1))
          ((1 *)) s-%offset%)
        (f2cl-lib:fref s-%data%
          ((f2cl-lib:int-mul 2 m)) ((1 *)) s-%offset%)
        (f2cl-lib:array-slice v-%data%
          f2cl-lib:complex16 (1 m) ((1 ldv) (1 *))
          v-%offset%))
      (setf alpha (f2cl-lib:fref v-%data% (1 m)
        ((1 ldv) (1 *)) v-%offset%))
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
        (zlarfg 3 alpha
          (f2cl-lib:array-slice v-%data%
            f2cl-lib:complex16 (2 m)
            ((1 ldv) (1 *)) v-%offset%)
          1
          (f2cl-lib:array-slice v-%data%
            f2cl-lib:complex16 (1 m)
            ((1 ldv) (1 *)) v-%offset%)))
        (declare (ignore var-0 var-2 var-3 var-4))
        (when var-1 (setf alpha var-1))))
    (t
      (setf beta
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
          h-%offset%)
        (setf (f2cl-lib:fref v-%data% (2 m)
          ((1 ldv) (1 *)) v-%offset%)

```

```

(f2cl-lib:fref h-%data%
  ((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *)))
h-%offset%)
(setf (f2cl-lib:fref v-%data% (3 m)
  ((1 ldv) (1 *)) v-%offset%)
(f2cl-lib:fref h-%data%
  ((f2cl-lib:int-add k 3) k) ((1 ldh) (1 *)))
h-%offset%)
(multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
  (zlarfg 3 beta
    (f2cl-lib:array-slice v-%data%
      f2cl-lib:complex16 (2 m)
      ((1 ldv) (1 *)) v-%offset%)
    1
    (f2cl-lib:array-slice v-%data%
      f2cl-lib:complex16 (1 m)
      ((1 ldv) (1 *)) v-%offset%)))
(declare (ignore var-0 var-2 var-3 var-4))
(when var-1 (setf beta var-1)))
(cond
  ((or
    (/= (f2cl-lib:fref h
      ((f2cl-lib:int-add k 3) k) ((1 ldh) (1 *)))
      zero)
    (/=
      (f2cl-lib:fref h
        ((f2cl-lib:int-add k 3) (f2cl-lib:int-add k 1))
        ((1 ldh) (1 *)))
      zero)
    (=
      (f2cl-lib:fref h
        ((f2cl-lib:int-add k 3) (f2cl-lib:int-add k 2))
        ((1 ldh) (1 *)))
      zero)))
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *)))
    h-%offset%)
  beta)
(setf
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *)))
  h-%offset%)
zero)
(setf
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 3) k) ((1 ldh) (1 *)))
  h-%offset%)
zero))
(t

```

```

(zlaqr1 3
  (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
    ((+ k 1) (f2cl-lib:int-add k 1))
    ((1 ldh) (1 *)) h-%offset%)
  ldh
  (f2cl-lib:fref s-%data%
    ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 m) 1))
    ((1 *)) s-%offset%)
  (f2cl-lib:fref s-%data%
    ((f2cl-lib:int-mul 2 m)) ((1 *)) s-%offset%)
  vt)
(setf alpha (f2cl-lib:fref vt (1) ((1 3))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4)
  (zlarfg 3 alpha
    (f2cl-lib:array-slice vt
      f2cl-lib:complex16 (2) ((1 3))) 1
    (f2cl-lib:array-slice vt
      f2cl-lib:complex16 (1) ((1 3))))
  (declare (ignore var-0 var-2 var-3 var-4))
  (when var-1 (setf alpha var-1)))
(setf refsum
  (coerce
    (* (f2cl-lib:dconjg
      (f2cl-lib:fref vt (1) ((1 3))))
      (+
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
          h-%offset%)
        (* (f2cl-lib:dconjg
          (f2cl-lib:fref vt (2) ((1 3))))
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 2) k)
            ((1 ldh) (1 *)) h-%offset%))))))
    'f2cl-lib:complex16))
(cond
  (>
    (+
      (cabs1
        (+ (f2cl-lib:fref h
          ((f2cl-lib:int-add k 2) k)
          ((1 ldh) (1 *)))
          (* -1 refsum (f2cl-lib:fref vt (2)
            ((1 3))))))
        (cabs1 (* refsum (f2cl-lib:fref vt (3)
          ((1 3))))))
      (* ulp
        (+ (cabs1 (f2cl-lib:fref h (k k)
          ((1 ldh) (1 *)))
          (cabs1

```

```

(f2cl-lib:fref h
  ((f2cl-lib:int-add k 1)
   (f2cl-lib:int-add k 1))
  ((1 ldh) (1 *))))
(cabs1
  (f2cl-lib:fref h
    ((f2cl-lib:int-add k 2)
     (f2cl-lib:int-add k 2))
    ((1 ldh) (1 *))))))
(setf
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
    h-%offset%)
  beta)
(setf
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *))
    h-%offset%)
  zero)
(setf
  (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 3) k) ((1 ldh) (1 *))
    h-%offset%)
  zero))
(t
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
      h-%offset%)
    (-
      (f2cl-lib:fref h-%data%
        ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
        h-%offset%)
      refsum))
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *))
      h-%offset%)
    zero)
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 3) k) ((1 ldh) (1 *))
      h-%offset%)
    zero)
  (setf (f2cl-lib:fref v-%data% (1 m)
    ((1 ldv) (1 *)) v-%offset%)
    (f2cl-lib:fref vt (1) ((1 3))))
  (setf (f2cl-lib:fref v-%data% (2 m)
    ((1 ldv) (1 *)) v-%offset%)
    (f2cl-lib:fref vt (2) ((1 3))))

```



```

      (setf (f2cl-lib:fref v-%data% (3 m)
        ((1 ldv) (1 *)) v-%offset%)
        (f2cl-lib:fref vt (3) ((1 3)))))))))
    label10))
  (setf k
    (f2cl-lib:int-add krcol (f2cl-lib:int-mul 3
      (f2cl-lib:int-sub m22 1))))
  (cond
    (bmp22
      (cond
        ((= k (f2cl-lib:int-add ktop (f2cl-lib:int-sub 1)))
          (zlaqr1 2
            (f2cl-lib:array-slice h-%data% f2cl-lib:complex16
              ((+ k 1) (f2cl-lib:int-add k 1))
              ((1 ldh) (1 *)) h-%offset%)
            ldh
            (f2cl-lib:fref s-%data%
              ((f2cl-lib:int-sub (f2cl-lib:int-mul 2 m22) 1))
              ((1 *)) s-%offset%)
            (f2cl-lib:fref s-%data%
              ((f2cl-lib:int-mul 2 m22)) ((1 *)) s-%offset%)
            (f2cl-lib:array-slice v-%data%
              f2cl-lib:complex16 (1 m22)
              ((1 ldv) (1 *)) v-%offset%))
          (setf beta (f2cl-lib:fref v-%data% (1 m22)
            ((1 ldv) (1 *)) v-%offset%))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
            (zlarfg 2 beta
              (f2cl-lib:array-slice v-%data%
                f2cl-lib:complex16 (2 m22)
                ((1 ldv) (1 *)) v-%offset%)
              1
              (f2cl-lib:array-slice v-%data%
                f2cl-lib:complex16 (1 m22)
                ((1 ldv) (1 *)) v-%offset%)))
            (declare (ignore var-0 var-2 var-3 var-4))
            (when var-1 (setf beta var-1))))
        (t
          (setf beta
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
              h-%offset%)
            (setf (f2cl-lib:fref v-%data% (2 m22)
              ((1 ldv) (1 *)) v-%offset%)
              (f2cl-lib:fref h-%data%
                ((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *))
                h-%offset%))
            (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
              (zlarfg 2 beta
                (f2cl-lib:array-slice v-%data%

```

```

f2cl-lib:complex16 (2 m22)
((1 ldv) (1 *)) v-%offset%)
1
(f2cl-lib:array-slice v-%data%
f2cl-lib:complex16 (1 m22)
((1 ldv) (1 *)) v-%offset%))
(declare (ignore var-0 var-2 var-3 var-4))
(when var-1 (setf beta var-1)))
(setf
(f2cl-lib:fref h-%data%
((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
h-%offset%)
beta)
(setf
(f2cl-lib:fref h-%data%
((f2cl-lib:int-add k 2) k) ((1 ldh) (1 *))
h-%offset%)
zero))))))
(cond
(accum
(setf jbot
(min (the f2cl-lib:integer4 ndcol)
(the f2cl-lib:integer4 kbot))))
(wantt (setf jbot n)) (t (setf jbot kbot)))
(f2cl-lib:fdo (j
(max (the f2cl-lib:integer4 ktop)
(the f2cl-lib:integer4 krcol))
(f2cl-lib:int-add j 1))
(> j jbot) nil)
(tagbody
(setf mend
(min mbot
(the f2cl-lib:integer4
(truncate (+ (- j krcol) 2) 3))))
(f2cl-lib:fdo (m mtop (f2cl-lib:int-add m 1))
(> m mend) nil)
(tagbody
(setf k
(f2cl-lib:int-add krcol
(f2cl-lib:int-mul 3 (f2cl-lib:int-sub m 1))))
(setf refsum
(coerce
(*
(f2cl-lib:dconjg
(f2cl-lib:fref v-%data% (1 m)
((1 ldv) (1 *)) v-%offset%))
+
(f2cl-lib:fref h-%data%
((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
h-%offset%)

```

```

      (*
      (f2c1-lib:dconjg
      (f2c1-lib:fref v-%data% (2 m)
      ((1 ldv) (1 *)) v-%offset%))
      (f2c1-lib:fref h-%data%
      ((f2c1-lib:int-add k 2) j) ((1 ldh) (1 *))
      h-%offset%))
      (*
      (f2c1-lib:dconjg
      (f2c1-lib:fref v-%data% (3 m)
      ((1 ldv) (1 *)) v-%offset%))
      (f2c1-lib:fref h-%data%
      ((f2c1-lib:int-add k 3) j) ((1 ldh) (1 *))
      h-%offset%)))
      'f2c1-lib:complex16))
    (setf
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 1) j) ((1 ldh) (1 *))
    h-%offset%)
    (-
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 1) j) ((1 ldh) (1 *))
    h-%offset%)
    refsum))
    (setf
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 2) j) ((1 ldh) (1 *))
    h-%offset%)
    (-
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 2) j) ((1 ldh) (1 *))
    h-%offset%)
    (* refsum (f2c1-lib:fref v-%data% (2 m)
    ((1 ldv) (1 *)) v-%offset%))))
    (setf
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 3) j) ((1 ldh) (1 *))
    h-%offset%)
    (-
    (f2c1-lib:fref h-%data%
    ((f2c1-lib:int-add k 3) j) ((1 ldh) (1 *))
    h-%offset%)
    (* refsum (f2c1-lib:fref v-%data% (3 m)
    ((1 ldv) (1 *)) v-%offset%))))
    label20))
  label30))
(cond
 (bmp22
  (setf k
    (f2c1-lib:int-add krcol

```

```

(f2cl-lib:int-mul 3 (f2cl-lib:int-sub m22 1)))
(f2cl-lib:fdo (j
  (max (the f2cl-lib:integer4 (f2cl-lib:int-add k 1))
    (the f2cl-lib:integer4 ktop))
  (f2cl-lib:int-add j 1))
(> j jbot) nil)
(tagbody
  (setf refsum
    (coerce
      (*
        (f2cl-lib:dconjg
          (f2cl-lib:fref v-%data% (1 m22)
            ((1 ldv) (1 *)) v-%offset%))
        (+
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
            h-%offset%)
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref v-%data% (2 m22)
                ((1 ldv) (1 *)) v-%offset%))
            (f2cl-lib:fref h-%data%
              ((f2cl-lib:int-add k 2) j) ((1 ldh) (1 *))
              h-%offset%))))))
    'f2cl-lib:complex16))
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
      h-%offset%)
    (-
      (f2cl-lib:fref h-%data%
        ((f2cl-lib:int-add k 1) j) ((1 ldh) (1 *))
        h-%offset%)
      refsum))
  (setf
    (f2cl-lib:fref h-%data%
      ((f2cl-lib:int-add k 2) j) ((1 ldh) (1 *))
      h-%offset%)
    (-
      (f2cl-lib:fref h-%data%
        ((f2cl-lib:int-add k 2) j) ((1 ldh) (1 *))
        h-%offset%)
      (* refsum
        (f2cl-lib:fref v-%data% (2 m22)
          ((1 ldv) (1 *)) v-%offset%))))
    label40))))
(cond
  (accum
    (setf jtop
      (max (the f2cl-lib:integer4 ktop)

```

```

        (the f2cl-lib:integer4 incol))))
    (wantt (setf jtop 1)) (t (setf jtop ktop)))
    (f2cl-lib:fdo (m mtop (f2cl-lib:int-add m 1))
  ((> m mbot) nil)
  (tagbody
    (cond
      ((/= (f2cl-lib:fref v (1 m) ((1 ldv) (1 *))) zero)
        (setf k
          (f2cl-lib:int-add krcol
            (f2cl-lib:int-mul 3 (f2cl-lib:int-sub m 1))))
        (f2cl-lib:fdo (j jtop (f2cl-lib:int-add j 1))
  ((> j
    (min (the f2cl-lib:integer4 kbot)
      (the f2cl-lib:integer4 (f2cl-lib:int-add k 3))))
    nil)
    (tagbody
      (setf refsum
        (* (f2cl-lib:fref v-%data% (1 m)
          ((1 ldv) (1 *)) v-%offset%)
          (+
            (f2cl-lib:fref h-%data% (j
              (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
              h-%offset%)
            (* (f2cl-lib:fref v-%data% (2 m)
              ((1 ldv) (1 *)) v-%offset%)
              (f2cl-lib:fref h-%data% (j
                (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
                h-%offset%))
            (* (f2cl-lib:fref v-%data% (3 m)
              ((1 ldv) (1 *)) v-%offset%)
              (f2cl-lib:fref h-%data% (j
                (f2cl-lib:int-add k 3)) ((1 ldh) (1 *))
                h-%offset%))))))
        (setf
          (f2cl-lib:fref h-%data% (j
            (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
            h-%offset%)
          (-
            (f2cl-lib:fref h-%data% (j
              (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
              h-%offset%)
            refsum))
        (setf
          (f2cl-lib:fref h-%data% (j
            (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
            h-%offset%)
          (-
            (f2cl-lib:fref h-%data% (j
              (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
              h-%offset%)

```

```

(* refsum
  (f2cl-lib:dconjg
    (f2cl-lib:fref v-%data% (2 m)
      ((1 ldv) (1 *)) v-%offset%))))
(setf
  (f2cl-lib:fref h-%data% (j
    (f2cl-lib:int-add k 3)) ((1 ldh) (1 *))
    h-%offset%)
  (-
    (f2cl-lib:fref h-%data% (j
      (f2cl-lib:int-add k 3)) ((1 ldh) (1 *))
      h-%offset%)
    (* refsum
      (f2cl-lib:dconjg
        (f2cl-lib:fref v-%data% (3 m)
          ((1 ldv) (1 *)) v-%offset%))))
    label50))
(cond
  (accum (setf kms (f2cl-lib:int-sub k incol))
    (f2cl-lib:fdo (j
      (max (the f2cl-lib:integer4 1)
        (the f2cl-lib:integer4
          (f2cl-lib:int-add ktop
            (f2cl-lib:int-sub incol))))
      (f2cl-lib:int-add j 1))
    (> j kdu) nil)
  (tagbody
    (setf refsum
      (* (f2cl-lib:fref v-%data% (1 m)
        ((1 ldv) (1 *)) v-%offset%)
        (+
          (f2cl-lib:fref u-%data% (j
            (f2cl-lib:int-add kms 1))
            ((1 ldu) (1 *)) u-%offset%)
          (* (f2cl-lib:fref v-%data% (2 m)
            ((1 ldv) (1 *)) v-%offset%)
            (f2cl-lib:fref u-%data% (j
              (f2cl-lib:int-add kms 2))
              ((1 ldu) (1 *)) u-%offset%)))
          (* (f2cl-lib:fref v-%data% (3 m)
            ((1 ldv) (1 *)) v-%offset%)
            (f2cl-lib:fref u-%data% (j
              (f2cl-lib:int-add kms 3))
              ((1 ldu) (1 *)) u-%offset%))))))
    (setf
      (f2cl-lib:fref u-%data% (j
        (f2cl-lib:int-add kms 1)) ((1 ldu) (1 *))
        u-%offset%)
      (-
        (f2cl-lib:fref u-%data% (j

```

```

        (f2cl-lib:int-add kms 1))
        ((1 ldu) (1 *)) u-%offset%)
    refsum))
(setf
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 2)) ((1 ldu) (1 *))
u-%offset%)
(-
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 2))
((1 ldu) (1 *)) u-%offset%)
(* refsum
(f2cl-lib:dconjg
(f2cl-lib:fref v-%data% (2 m)
((1 ldv) (1 *)) v-%offset%))))))
(setf
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 3)) ((1 ldu) (1 *))
u-%offset%)
(-
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 3))
((1 ldu) (1 *)) u-%offset%)
(* refsum
(f2cl-lib:dconjg
(f2cl-lib:fref v-%data% (3 m) ((1 ldv)
(1 *)) v-%offset%))))))
label60)))
(wantz
(f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
(> j ihiz) nil)
(tagbody
(setf refsum
(* (f2cl-lib:fref v-%data% (1 m)
((1 ldv) (1 *)) v-%offset%)
(+
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
z-%offset%)
(* (f2cl-lib:fref v-%data% (2 m)
((1 ldv) (1 *)) v-%offset%)
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 2))
((1 ldz) (1 *)) z-%offset%))
(* (f2cl-lib:fref v-%data% (3 m)
((1 ldv) (1 *)) v-%offset%)
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 3))
((1 ldz) (1 *)) z-%offset%))))))
(setf

```

```

(f2cl-lib:fref z-%data% (j
  (f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
  z-%offset%)
(-
  (f2cl-lib:fref z-%data% (j
    (f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
    z-%offset%)
  refsum))
(setf
  (f2cl-lib:fref z-%data% (j
    (f2cl-lib:int-add k 2)) ((1 ldz) (1 *))
    z-%offset%)
  (-
    (f2cl-lib:fref z-%data% (j
      (f2cl-lib:int-add k 2)) ((1 ldz) (1 *))
      z-%offset%)
    (* refsum
      (f2cl-lib:dconjg
        (f2cl-lib:fref v-%data% (2 m)
          ((1 ldv) (1 *)) v-%offset%))))))
(setf
  (f2cl-lib:fref z-%data% (j
    (f2cl-lib:int-add k 3)) ((1 ldz) (1 *))
    z-%offset%)
  (-
    (f2cl-lib:fref z-%data% (j
      (f2cl-lib:int-add k 3)) ((1 ldz) (1 *))
      z-%offset%)
    (* refsum
      (f2cl-lib:dconjg
        (f2cl-lib:fref v-%data% (3 m)
          ((1 ldv) (1 *)) v-%offset%))))))
  label170))))))
  label180))
(setf k
  (f2cl-lib:int-add krcol
    (f2cl-lib:int-mul 3 (f2cl-lib:int-sub m22 1))))
(cond
  (bmp22
    (cond
      ((/= (f2cl-lib:fref v (1 m22) ((1 ldv) (1 *))) zero)
        (f2cl-lib:fdo (j jtop (f2cl-lib:int-add j 1))
          (> j
            (min (the f2cl-lib:integer4 kbot)
              (the f2cl-lib:integer4
                (f2cl-lib:int-add k 3))))
            nil)
        (tagbody
          (setf refsum
            (* (f2cl-lib:fref v-%data% (1 m22)

```



```

((1 ldv) (1 *)) v-%offset%)
(
  (f2cl-lib:fref h-%data% (j
    (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%)
  (* (f2cl-lib:fref v-%data% (2 m22)
    ((1 ldv) (1 *)) v-%offset%)
    (f2cl-lib:fref h-%data% (j
      (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
      h-%offset%))))))
(setf
  (f2cl-lib:fref h-%data% (j
    (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%)
  (-
    (f2cl-lib:fref h-%data% (j
      (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
      h-%offset%)
    refsum))
(setf
  (f2cl-lib:fref h-%data% (j
    (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
    h-%offset%)
  (-
    (f2cl-lib:fref h-%data% (j
      (f2cl-lib:int-add k 2)) ((1 ldh) (1 *))
      h-%offset%)
    (* refsum
      (f2cl-lib:dconjg
        (f2cl-lib:fref v-%data% (2 m22)
          ((1 ldv) (1 *)) v-%offset%))))))
  label90))
(cond
  (accum (setf kms (f2cl-lib:int-sub k incol))
    (f2cl-lib:fdo (j
      (max (the f2cl-lib:integer4 1)
        (the f2cl-lib:integer4
          (f2cl-lib:int-add ktop
            (f2cl-lib:int-sub incol))))
      (f2cl-lib:int-add j 1))
    (> j kdu) nil)
  (tagbody
    (setf refsum
      (* (f2cl-lib:fref v-%data% (1 m22)
        ((1 ldv) (1 *)) v-%offset%)
        (+
          (f2cl-lib:fref u-%data% (j
            (f2cl-lib:int-add kms 1))
            ((1 ldu) (1 *)) u-%offset%)
          (* (f2cl-lib:fref v-%data% (2 m22)

```

```

((1 ldv) (1 *)) v-%offset%)
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 2))
((1 ldu) (1 *)) u-%offset%))))))
(setf
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 1))
((1 ldu) (1 *)) u-%offset%)
(-
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 1))
((1 ldu) (1 *)) u-%offset%)
refsum))
(setf
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 2))
((1 ldu) (1 *)) u-%offset%)
(-
(f2cl-lib:fref u-%data% (j
(f2cl-lib:int-add kms 2))
((1 ldu) (1 *)) u-%offset%)
(* refsum
(f2cl-lib:dconjg
(f2cl-lib:fref v-%data% (2 m22)
((1 ldv) (1 *)) v-%offset%))))))
label100)))
(wantz
(f2cl-lib:fdo (j iloz (f2cl-lib:int-add j 1))
(> j ihiz) nil)
(tagbody
(setf refsum
(* (f2cl-lib:fref v-%data% (1 m22)
((1 ldv) (1 *)) v-%offset%)
(+
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 1))
((1 ldz) (1 *)) z-%offset%)
(* (f2cl-lib:fref v-%data% (2 m22)
((1 ldv) (1 *)) v-%offset%)
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 2))
((1 ldz) (1 *)) z-%offset%))))))
(setf
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
z-%offset%)
(-
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 1)) ((1 ldz) (1 *))
z-%offset%)

```

```
(refsum))
(setf
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 2)) ((1 ldz) (1 *))
z-%offset%)
(-
(f2cl-lib:fref z-%data% (j
(f2cl-lib:int-add k 2)) ((1 ldz) (1 *))
z-%offset%)
(* refsum
(f2cl-lib:dconjg
(f2cl-lib:fref v-%data% (2 m22)
((1 ldv) (1 *)) v-%offset%))))))
label110)))))))
(setf mstart mtop)
(if
(<
(f2cl-lib:int-add krcol
(f2cl-lib:int-mul 3 (f2cl-lib:int-sub mstart 1)))
ktop)
(setf mstart (f2cl-lib:int-add mstart 1)))
(setf mend mbot)
(if bmp22 (setf mend (f2cl-lib:int-add mend 1)))
(if (= krcol (f2cl-lib:int-sub kbot 2))
(setf mend (f2cl-lib:int-add mend 1)))
(f2cl-lib:fdo (m mstart (f2cl-lib:int-add m 1))
((> m mend) nil)
(tagbody
(setf k
(min (the f2cl-lib:integer4 (f2cl-lib:int-sub kbot 1))
(the f2cl-lib:integer4
(f2cl-lib:int-add krcol
(f2cl-lib:int-mul 3 (f2cl-lib:int-sub m 1))))))
(cond
(/= (f2cl-lib:fref h ((f2cl-lib:int-add k 1) k)
((1 ldh) (1 *))) zero)
(setf tst1
(+ (cabs1 (f2cl-lib:fref h-%data% (k k)
((1 ldh) (1 *)) h-%offset%))
(cabs1
(f2cl-lib:fref h-%data%
((f2cl-lib:int-add k 1)
(f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
h-%offset%))))))
(cond
(= tst1 rzero)
(if (>= k (f2cl-lib:int-add ktop 1))
(setf tst1
(+ tst1
(cabs1
```

```

      (f2cl-lib:fref h-%data% (k
        (f2cl-lib:int-sub k 1)) ((1 ldh) (1 *))
        h-%offset%))))))
(if (>= k (f2cl-lib:int-add ktop 2))
  (setf tst1
    (+ tst1
      (cabs1
        (f2cl-lib:fref h-%data% (k
          (f2cl-lib:int-sub k 2)) ((1 ldh) (1 *))
          h-%offset%))))))
(if (>= k (f2cl-lib:int-add ktop 3))
  (setf tst1
    (+ tst1
      (cabs1
        (f2cl-lib:fref h-%data% (k
          (f2cl-lib:int-sub k 3)) ((1 ldh) (1 *))
          h-%offset%))))))
(if (<= k (f2cl-lib:int-sub kbot 2))
  (setf tst1
    (+ tst1
      (cabs1
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 2)
           (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
          h-%offset%))))))
(if (<= k (f2cl-lib:int-sub kbot 3))
  (setf tst1
    (+ tst1
      (cabs1
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 3)
           (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
          h-%offset%))))))
(if (<= k (f2cl-lib:int-sub kbot 4))
  (setf tst1
    (+ tst1
      (cabs1
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 4)
           (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
          h-%offset%))))))
(cond
  ((<=
    (cabs1 (f2cl-lib:fref h
      ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))))
    (max smlnum (* ulp tst1)))
    (setf h12
      (max
        (cabs1
          (f2cl-lib:fref h-%data%

```

```

      ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
      h-%offset%))
(cabs1
 (f2cl-lib:fref h-%data% (k
   (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
   h-%offset%))))
(setf h21
 (min
  (cabs1
   (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
    h-%offset%))
  (cabs1
   (f2cl-lib:fref h-%data% (k
    (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%))))))
(setf h11
 (max
  (cabs1
   (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1)
     (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%))
  (cabs1
   (- (f2cl-lib:fref h-%data% (k k)
    ((1 ldh) (1 *)) h-%offset%)
      (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1)
     (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%))))))
(setf h22
 (min
  (cabs1
   (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1)
     (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%))
  (cabs1
   (- (f2cl-lib:fref h-%data% (k k)
    ((1 ldh) (1 *)) h-%offset%)
      (f2cl-lib:fref h-%data%
    ((f2cl-lib:int-add k 1)
     (f2cl-lib:int-add k 1)) ((1 ldh) (1 *))
    h-%offset%))))))
(setf scl (+ h11 h12))
(setf tst2 (* h22 (/ h11 scl)))
(if
 (or (= tst2 rzero)
      (<= (* h21 (/ h12 scl))
           (max smlnum (* ulp tst2))))

```

```

      (setf
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 1) k) ((1 ldh) (1 *))
          h-%offset%)
        zero))))))
    label120))
  (setf mnd
    (min nbmps
      (the f2cl-lib:integer4 (truncate (- kbot krcol 1) 3))))
    (f2cl-lib:fdo (m mtop (f2cl-lib:int-add m 1))
      (> m mnd) nil)
    (tagbody
      (setf k
        (f2cl-lib:int-add krcol
          (f2cl-lib:int-mul 3 (f2cl-lib:int-sub m 1))))
      (setf refsum
        (* (f2cl-lib:fref v-%data% (1 m) ((1 ldv) (1 *))
          v-%offset%)
          (f2cl-lib:fref v-%data% (3 m)
            ((1 ldv) (1 *)) v-%offset%)
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 4) (f2cl-lib:int-add k 3))
            ((1 ldh) (1 *)) h-%offset%)))
      (setf
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 4) (f2cl-lib:int-add k 1))
          ((1 ldh) (1 *)) h-%offset%)
        (- refsum))
      (setf
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 4) (f2cl-lib:int-add k 2))
          ((1 ldh) (1 *)) h-%offset%)
        (coerce
          (* (- refsum)
            (f2cl-lib:dconjg
              (f2cl-lib:fref v-%data% (2 m)
                ((1 ldv) (1 *)) v-%offset%)))
          'f2cl-lib:complex16))
      (setf
        (f2cl-lib:fref h-%data%
          ((f2cl-lib:int-add k 4) (f2cl-lib:int-add k 3))
          ((1 ldh) (1 *)) h-%offset%)
        (-
          (f2cl-lib:fref h-%data%
            ((f2cl-lib:int-add k 4) (f2cl-lib:int-add k 3))
            ((1 ldh) (1 *)) h-%offset%)
          (* refsum
            (f2cl-lib:dconjg
              (f2cl-lib:fref v-%data% (3 m)
                ((1 ldv) (1 *)) v-%offset%))))))

```

```

        label130))
    label140))
(cond
  (accum
    (cond (wantt (setf jtop 1) (setf jbot n))
      (t (setf jtop ktop) (setf jbot kbot)))
    (cond
      ((or (not blk22) (< incol ktop)
        (> ndcol kbot) (<= ns 2))
        (setf k1
          (max (the f2cl-lib:integer4 1)
            (the f2cl-lib:integer4
              (f2cl-lib:int-sub ktop incol))))
          (setf nu
            (f2cl-lib:int-add
              (f2cl-lib:int-sub kdu
                (max (the f2cl-lib:integer4 0)
                  (the f2cl-lib:integer4
                    (f2cl-lib:int-sub ndcol kbot)))
                k1)
              1))
            (f2cl-lib:fdo (jcol
              (f2cl-lib:int-add
                (min (the f2cl-lib:integer4 ndcol)
                  (the f2cl-lib:integer4 kbot)) 1)
                (f2cl-lib:int-add jcol nh))
                (> jcol jbot) nil)
              (tagbody
                (setf jlen
                  (min (the f2cl-lib:integer4 nh)
                    (the f2cl-lib:integer4
                      (f2cl-lib:int-add
                        (f2cl-lib:int-sub jbot jcol) 1))))
                  (multiple-value-bind
                    (var-0 var-1 var-2 var-3 var-4 var-5
                     var-6 var-7 var-8 var-9 var-10
                     var-11 var-12)
                    (zgemm "C" "N" nu jlen nu one
                      (f2cl-lib:array-slice u-%data%
                        f2cl-lib:complex16 (k1 k1)
                        ((1 ldu) (1 *)) u-%offset%)
                        ldu
                        (f2cl-lib:array-slice h-%data%
                          f2cl-lib:complex16 ((+ incol k1) jcol)
                          ((1 ldh) (1 *)) h-%offset%)
                          ldh zero wh ldwh)
                    (declare (ignore var-0 var-1 var-6 var-8
                      var-11))
                    (when var-2 (setf nu var-2))
                    (when var-3 (setf jlen var-3))

```

```

      (when var-4 (setf nu var-4))
      (when var-5 (setf one var-5))
      (when var-7 (setf ldu var-7))
      (when var-9 (setf ldh var-9))
      (when var-10 (setf zero var-10))
      (when var-12 (setf ldwh var-12)))
      (zlacpy "ALL" nu jlen wh ldwh
      (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16 ((+ incol k1) jcol)
      ((1 ldh) (1 *)) h-%offset%)
      ldh)
      label1150))
      (f2cl-lib:fdo (jrow jtop (f2cl-lib:int-add jrow nv))
      (> jrow
      (f2cl-lib:int-add
      (max (the f2cl-lib:integer4 ktop)
      (the f2cl-lib:integer4 incol))
      (f2cl-lib:int-sub 1)))
      nil)
      (tagbody
      (setf jlen
      (min (the f2cl-lib:integer4 nv)
      (the f2cl-lib:integer4
      (f2cl-lib:int-sub
      (max (the f2cl-lib:integer4 ktop)
      (the f2cl-lib:integer4 incol))
      jrow))))
      (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
      var-7 var-8 var-9 var-10
      var-11 var-12)
      (zgemm "N" "N" jlen nu nu one
      (f2cl-lib:array-slice h-%data%
      f2cl-lib:complex16
      (jrow (f2cl-lib:int-add incol k1))
      ((1 ldh) (1 *)) h-%offset%)
      ldh
      (f2cl-lib:array-slice u-%data%
      f2cl-lib:complex16 (k1 k1)
      ((1 ldu) (1 *)) u-%offset%)
      ldu zero ww ldwv)
      (declare (ignore var-0 var-1 var-6
      var-8 var-11))
      (when var-2 (setf jlen var-2))
      (when var-3 (setf nu var-3))
      (when var-4 (setf nu var-4))
      (when var-5 (setf one var-5))
      (when var-7 (setf ldh var-7))
      (when var-9 (setf ldu var-9))
      (when var-10 (setf zero var-10))

```



```

        (when var-12 (setf ldwv var-12)))
      (zlacpy "ALL" jlen nu wv ldwv
        (f2cl-lib:array-slice h-%data%
          f2cl-lib:complex16
          (jrow (f2cl-lib:int-add incol k1))
          ((1 ldh) (1 *)) h-%offset%)
        ldh)
      label1160))
    (cond
      (wantz
        (f2cl-lib:fdo (jrow iloz (f2cl-lib:int-add jrow nv))
          ((> jrow ihiz)
            nil)
          (tagbody
            (setf jlen
              (min (the f2cl-lib:integer4 nv)
                (the f2cl-lib:integer4
                  (f2cl-lib:int-add
                    (f2cl-lib:int-sub ihiz jrow) 1))))))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8 var-9 var-10
                var-11 var-12)
              (zgemm "N" "N" jlen nu nu one
                (f2cl-lib:array-slice z-%data%
                  f2cl-lib:complex16
                  (jrow (f2cl-lib:int-add incol k1))
                  ((1 ldz) (1 *)) z-%offset%)
                ldz
                (f2cl-lib:array-slice u-%data%
                  f2cl-lib:complex16 (k1 k1)
                  ((1 ldu) (1 *)) u-%offset%)
                ldu zero wv ldwv)
              (declare (ignore var-0 var-1 var-6
                var-8 var-11))
              (when var-2 (setf jlen var-2))
              (when var-3 (setf nu var-3))
              (when var-4 (setf nu var-4))
              (when var-5 (setf one var-5))
              (when var-7 (setf ldz var-7))
              (when var-9 (setf ldu var-9))
              (when var-10 (setf zero var-10))
              (when var-12 (setf ldwv var-12)))
            (zlacpy "ALL" jlen nu wv ldwv
              (f2cl-lib:array-slice z-%data%
                f2cl-lib:complex16
                (jrow (f2cl-lib:int-add incol k1))
                ((1 ldz) (1 *)) z-%offset%)
              ldz)
            label1170))))))

```

```

(t (setf i2 (the f2cl-lib:integer4
              (truncate (+ kdu 1) 2)))
  (setf i4 kdu)
  (setf j2 (f2cl-lib:int-sub i4 i2))
  (setf j4 kdu)
  (setf kzs (f2cl-lib:int-sub j4 j2
                          (f2cl-lib:int-add ns 1)))
  (setf knz (f2cl-lib:int-add ns 1))
  (f2cl-lib:fdo (jcol
                (f2cl-lib:int-add
                  (min (the f2cl-lib:integer4 ndcol)
                        (the f2cl-lib:integer4 kbot)) 1)
                (f2cl-lib:int-add jcol nh))
    (> jcol jbot) nil)
  (tagbody
    (setf jlen
      (min (the f2cl-lib:integer4 nh)
            (the f2cl-lib:integer4
              (f2cl-lib:int-add
                (f2cl-lib:int-sub jbot jcol) 1))))
    (zlacpy "ALL" knz jlen
      (f2cl-lib:array-slice h-%data%
        f2cl-lib:complex16
        ((+ incol 1 j2) jcol) ((1 ldh) (1 *))
        h-%offset%)
      ldh
      (f2cl-lib:array-slice wh-%data%
        f2cl-lib:complex16 ((+ kzs 1) 1)
        ((1 ldwh) (1 *)) wh-%offset%)
      ldwh)
    (multiple-value-bind (var-0 var-1 var-2
                          var-3 var-4 var-5 var-6)
      (zlaset "ALL" kzs jlen zero zero wh ldwh)
      (declare (ignore var-0 var-5))
      (when var-1 (setf kzs var-1))
      (when var-2 (setf jlen var-2))
      (when var-3 (setf zero var-3))
      (when var-4 (setf zero var-4))
      (when var-6 (setf ldwh var-6)))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5
       var-6 var-7 var-8 var-9 var-10)
      (ztrmm "L" "U" "C" "N" knz jlen one
        (f2cl-lib:array-slice u-%data%
          f2cl-lib:complex16
          ((+ j2 1) (f2cl-lib:int-add 1 kzs))
          ((1 ldu) (1 *)) u-%offset%)
        ldu
        (f2cl-lib:array-slice wh-%data%
          f2cl-lib:complex16 ((+ kzs 1) 1)

```

```

      ((1 ldwh) (1 *)) wh-%offset%)
    ldwh)
  (declare (ignore var-0 var-1 var-2 var-3
             var-7 var-9))
  (when var-4 (setf knz var-4))
  (when var-5 (setf jlen var-5))
  (when var-6 (setf one var-6))
  (when var-8 (setf ldu var-8))
  (when var-10 (setf ldwh var-10)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10
  var-11 var-12)
 (zgemm "C" "N" i2 jlen j2 one u ldu
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16 ((+ incol 1) jcol)
    ((1 ldh) (1 *)) h-%offset%)
  ldh one wh ldwh)
 (declare (ignore var-0 var-1 var-6
                  var-8 var-11))
 (when var-2 (setf i2 var-2))
 (when var-3 (setf jlen var-3))
 (when var-4 (setf j2 var-4))
 (when var-5 (setf one var-5))
 (when var-7 (setf ldu var-7))
 (when var-9 (setf ldh var-9))
 (when var-10 (setf one var-10))
 (when var-12 (setf ldwh var-12)))
(zlacpy "ALL" j2 jlen
 (f2cl-lib:array-slice h-%data%
  f2cl-lib:complex16 ((+ incol 1) jcol)
  ((1 ldh) (1 *)) h-%offset%)
ldh
(f2cl-lib:array-slice wh-%data%
 f2cl-lib:complex16 ((+ i2 1) 1)
 ((1 ldwh) (1 *)) wh-%offset%)
ldwh)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6
  var-7 var-8 var-9 var-10)
 (ztrmm "L" "L" "C" "N" j2 jlen one
  (f2cl-lib:array-slice u-%data%
    f2cl-lib:complex16
    (1 (f2cl-lib:int-add i2 1))
    ((1 ldu) (1 *)) u-%offset%)
  ldu
  (f2cl-lib:array-slice wh-%data%
    f2cl-lib:complex16 ((+ i2 1) 1)
    ((1 ldwh) (1 *)) wh-%offset%)
  ldwh)

```

```

(declare (ignore var-0 var-1 var-2 var-3
           var-7 var-9))
(when var-4 (setf j2 var-4))
(when var-5 (setf jlen var-5))
(when var-6 (setf one var-6))
(when var-8 (setf ldu var-8))
(when var-10 (setf ldwh var-10)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6
  var-7 var-8 var-9 var-10
  var-11 var-12)
 (zgemm "C" "N" (f2cl-lib:int-sub i4 i2)
  jlen (f2cl-lib:int-sub j4 j2)
  one
  (f2cl-lib:array-slice u-%data%
   f2cl-lib:complex16
   ((+ j2 1) (f2cl-lib:int-add i2 1))
   ((1 ldu) (1 *)) u-%offset%)
  ldu
  (f2cl-lib:array-slice h-%data%
   f2cl-lib:complex16
   ((+ incol 1 j2) jcol) ((1 ldh) (1 *))
   h-%offset%)
  ldh one
  (f2cl-lib:array-slice wh-%data%
   f2cl-lib:complex16 ((+ i2 1) 1)
   ((1 ldwh) (1 *)) wh-%offset%)
  ldwh)
(declare (ignore var-0 var-1 var-2 var-4
           var-6 var-8 var-11))
(when var-3 (setf jlen var-3))
(when var-5 (setf one var-5))
(when var-7 (setf ldu var-7))
(when var-9 (setf ldh var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldwh var-12)))
(zlacpy "ALL" kdu jlen wh ldwh
 (f2cl-lib:array-slice h-%data%
  f2cl-lib:complex16 ((+ incol 1) jcol)
  ((1 ldh) (1 *)) h-%offset%)
 ldh)
label180))
(f2cl-lib:fdo (jrow jtop (f2cl-lib:int-add jrow nv))
 (> jrow
  (f2cl-lib:int-add
   (max (the f2cl-lib:integer4 incol)
    (the f2cl-lib:integer4 ktop))
   (f2cl-lib:int-sub 1)))
 nil)
(tagbody

```

```

(setf jlen
  (min (the f2cl-lib:integer4 nv)
        (the f2cl-lib:integer4
          (f2cl-lib:int-sub
            (max (the f2cl-lib:integer4 incol)
                  (the f2cl-lib:integer4 ktop))
            jrow))))
(zlacpy "ALL" jlen knz
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16
    (jrow (f2cl-lib:int-add incol 1 j2))
    ((1 ldh) (1 *)) h-%offset%)
  ldh
  (f2cl-lib:array-slice wv-%data%
    f2cl-lib:complex16
    (1 (f2cl-lib:int-add 1 kzs))
    ((1 ldwv) (1 *)) wv-%offset%)
  ldwv)
(multiple-value-bind (var-0 var-1 var-2 var-3
  var-4 var-5 var-6)
  (zlaset "ALL" jlen kzs zero zero wv ldwv)
  (declare (ignore var-0 var-5))
  (when var-1 (setf jlen var-1))
  (when var-2 (setf kzs var-2))
  (when var-3 (setf zero var-3))
  (when var-4 (setf zero var-4))
  (when var-6 (setf ldwv var-6)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5
   var-6 var-7 var-8 var-9 var-10)
  (ztrmm "R" "U" "N" "N" jlen knz one
    (f2cl-lib:array-slice u-%data%
      f2cl-lib:complex16
      ((+ j2 1) (f2cl-lib:int-add 1 kzs))
      ((1 ldu) (1 *)) u-%offset%)
    ldu
    (f2cl-lib:array-slice wv-%data%
      f2cl-lib:complex16
      (1 (f2cl-lib:int-add 1 kzs))
      ((1 ldwv) (1 *)) wv-%offset%)
    ldwv)
  (declare (ignore var-0 var-1 var-2 var-3
    var-7 var-9))
  (when var-4 (setf jlen var-4))
  (when var-5 (setf knz var-5))
  (when var-6 (setf one var-6))
  (when var-8 (setf ldu var-8))
  (when var-10 (setf ldwv var-10)))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6

```

```

var-7 var-8 var-9 var-10
var-11 var-12)
(zgemm "N" "N" jlen i2 j2 one
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16
    (jrow (f2cl-lib:int-add incol 1))
      ((1 ldh) (1 *)) h-%offset%)
  ldh u ldu one wv ldwv)
(declare (ignore var-0 var-1 var-6
  var-8 var-11))
(when var-2 (setf jlen var-2))
(when var-3 (setf i2 var-3))
(when var-4 (setf j2 var-4))
(when var-5 (setf one var-5))
(when var-7 (setf ldh var-7))
(when var-9 (setf ldu var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldwv var-12)))
(zlacpy "ALL" jlen j2
  (f2cl-lib:array-slice h-%data%
    f2cl-lib:complex16
    (jrow (f2cl-lib:int-add incol 1))
      ((1 ldh) (1 *)) h-%offset%)
  ldh
  (f2cl-lib:array-slice wv-%data%
    f2cl-lib:complex16
    (1 (f2cl-lib:int-add 1 i2))
      ((1 ldwv) (1 *)) wv-%offset%)
  ldwv)
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6
    var-7 var-8 var-9 var-10)
  (ztrmm "R" "L" "N" "N" jlen
    (f2cl-lib:int-sub i4 i2) one
    (f2cl-lib:array-slice u-%data%
      f2cl-lib:complex16
      (1 (f2cl-lib:int-add i2 1))
        ((1 ldu) (1 *)) u-%offset%)
    ldu
    (f2cl-lib:array-slice wv-%data%
      f2cl-lib:complex16
      (1 (f2cl-lib:int-add 1 i2))
        ((1 ldwv) (1 *)) wv-%offset%)
    ldwv)
  (declare (ignore var-0 var-1 var-2 var-3
    var-5 var-7 var-9))
  (when var-4 (setf jlen var-4))
  (when var-6 (setf one var-6))
  (when var-8 (setf ldu var-8))
  (when var-10 (setf ldwv var-10)))

```

```

(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10
  var-11 var-12)
 (zgemm "N" "N" jlen
  (f2cl-lib:int-sub i4 i2)
  (f2cl-lib:int-sub j4 j2)
  one
  (f2cl-lib:array-slice h-%data%
   f2cl-lib:complex16
   (jrow (f2cl-lib:int-add incol 1 j2))
   ((1 ldh) (1 *)) h-%offset%)
  ldh
  (f2cl-lib:array-slice u-%data%
   f2cl-lib:complex16
   ((+ j2 1) (f2cl-lib:int-add i2 1))
   ((1 ldu) (1 *)) u-%offset%)
  ldu one
  (f2cl-lib:array-slice wv-%data%
   f2cl-lib:complex16
   (1 (f2cl-lib:int-add 1 i2)) ((1 ldwv)
    (1 *)) wv-%offset%)
  ldwv)
 (declare (ignore var-0 var-1 var-3 var-4
  var-6 var-8 var-11))
 (when var-2 (setf jlen var-2))
 (when var-5 (setf one var-5))
 (when var-7 (setf ldh var-7))
 (when var-9 (setf ldu var-9))
 (when var-10 (setf one var-10))
 (when var-12 (setf ldwv var-12)))
 (zlacpy "ALL" jlen kdu wv ldwv
  (f2cl-lib:array-slice h-%data%
   f2cl-lib:complex16
   (jrow (f2cl-lib:int-add incol 1)) ((1 ldh)
    (1 *)) h-%offset%)
  ldh)
 label190))
(cond
 (wantz
  (f2cl-lib:fdo (jrow iloz (f2cl-lib:int-add jrow nv))
  (> jrow ihiz)
  nil)
 (tagbody
  (setf jlen
   (min (the f2cl-lib:integer4 nv)
    (the f2cl-lib:integer4
     (f2cl-lib:int-add
      (f2cl-lib:int-sub ihiz jrow) 1))))
  (zlacpy "ALL" jlen knz

```

```

(f2cl-lib:array-slice z-%data%
 f2cl-lib:complex16
 (jrow (f2cl-lib:int-add incol 1 j2))
 ((1 ldz) (1 *)) z-%offset%)
ldz
(f2cl-lib:array-slice wv-%data%
 f2cl-lib:complex16
 (1 (f2cl-lib:int-add 1 kzs))
 ((1 ldwv) (1 *)) wv-%offset%)
ldwv
(multiple-value-bind (var-0 var-1 var-2
 var-3 var-4 var-5 var-6)
 (zlaset "ALL" jlen kzs zero zero wv ldwv)
 (declare (ignore var-0 var-5))
 (when var-1 (setf jlen var-1))
 (when var-2 (setf kzs var-2))
 (when var-3 (setf zero var-3))
 (when var-4 (setf zero var-4))
 (when var-6 (setf ldwv var-6)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
 var-6 var-7 var-8 var-9 var-10)
 (ztrmm "R" "U" "N" "N" jlen knz one
 (f2cl-lib:array-slice u-%data%
 f2cl-lib:complex16
 ((+ j2 1) (f2cl-lib:int-add 1 kzs))
 ((1 ldu) (1 *)) u-%offset%)
 ldu
 (f2cl-lib:array-slice wv-%data%
 f2cl-lib:complex16
 (1 (f2cl-lib:int-add 1 kzs))
 ((1 ldwv) (1 *)) wv-%offset%)
 ldwv)
(declare (ignore
 var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf jlen var-4))
(when var-5 (setf knz var-5))
(when var-6 (setf one var-6))
(when var-8 (setf ldu var-8))
(when var-10 (setf ldwv var-10)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
 var-6 var-7 var-8 var-9 var-10
 var-11 var-12)
 (zgemm "N" "N" jlen i2 j2 one
 (f2cl-lib:array-slice z-%data%
 f2cl-lib:complex16
 (jrow (f2cl-lib:int-add incol 1))
 ((1 ldz) (1 *)) z-%offset%)
 ldz u ldu one wv ldwv)

```



```

(declare (ignore var-0 var-1 var-6
           var-8 var-11))
(when var-2 (setf jlen var-2))
(when var-3 (setf i2 var-3))
(when var-4 (setf j2 var-4))
(when var-5 (setf one var-5))
(when var-7 (setf ldz var-7))
(when var-9 (setf ldu var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldwv var-12)))
(zlacpy "ALL" jlen j2
 (f2cl-lib:array-slice z-%data%
  f2cl-lib:complex16
  (jrow (f2cl-lib:int-add incol 1))
   ((1 ldz) (1 *)) z-%offset%)
ldz
(f2cl-lib:array-slice wv-%data%
 f2cl-lib:complex16
 (1 (f2cl-lib:int-add 1 i2))
 ((1 ldwv) (1 *)) wv-%offset%)
ldwv)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10)
 (ztrmm "R" "L" "N" "N" jlen
  (f2cl-lib:int-sub i4 i2) one
  (f2cl-lib:array-slice u-%data%
   f2cl-lib:complex16
   (1 (f2cl-lib:int-add i2 1))
    ((1 ldu) (1 *)) u-%offset%)
  ldu
  (f2cl-lib:array-slice wv-%data%
   f2cl-lib:complex16
   (1 (f2cl-lib:int-add 1 i2))
    ((1 ldwv) (1 *)) wv-%offset%)
  ldwv)
(declare (ignore var-0 var-1 var-2 var-3
                 var-5 var-7 var-9))
(when var-4 (setf jlen var-4))
(when var-6 (setf one var-6))
(when var-8 (setf ldu var-8))
(when var-10 (setf ldwv var-10)))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10
  var-11 var-12)
 (zgemm "N" "N" jlen (f2cl-lib:int-sub i4 i2)
  (f2cl-lib:int-sub j4 j2) one
  (f2cl-lib:array-slice z-%data%
   f2cl-lib:complex16

```

```

(jrow (f2cl-lib:int-add incol 1 j2))
      ((1 ldz) (1 *)) z-%offset%)
ldz
(f2cl-lib:array-slice u-%data%
 f2cl-lib:complex16
 ((+ j2 1) (f2cl-lib:int-add i2 1))
 ((1 ldu) (1 *)) u-%offset%)
ldu one
(f2cl-lib:array-slice wv-%data%
 f2cl-lib:complex16
 (1 (f2cl-lib:int-add 1 i2)) ((1 ldwv)
 (1 *)) wv-%offset%)
ldwv)
(declare (ignore var-0 var-1 var-3 var-4
              var-6 var-8 var-11))
(when var-2 (setf jlen var-2))
(when var-5 (setf one var-5))
(when var-7 (setf ldz var-7))
(when var-9 (setf ldu var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldwv var-12)))
(zlacpy "ALL" jlen kdu wv ldwv
 (f2cl-lib:array-slice
  z-%data% f2cl-lib:complex16
  (jrow (f2cl-lib:int-add incol 1))
  ((1 ldz) (1 *)) z-%offset%)
 ldz)
label200)))))))))
label210))
end_label
(return
 (values nil nil nil nil nil nil nil nil ldh nil nil nil
  ldz nil nil nil
  ldu nil nil ldwv nil nil ldwh))))))

```

zlarfb LAPACK

— zlarfb.input —

```

)set break resume
)sys rm -f zlarfb.output
)spool zlarfb.output
)set message test on
)set message auto off
)clear all

```

```
)spool
)lisp (bye)
```

— zlarfb.help —

```
=====
zlarfb examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```
SUBROUTINE ZLARFB( SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV,
                  T, LDT, C, LDC, WORK, LDWORK )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          DIRECT, SIDE, STOREV, TRANS
```

```
INTEGER            K, LDC, LDT, LDV, LDWORK, M, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16         C( LDC, * ), T( LDT, * ), V( LDV, * ),
```

```
$                  WORK( LDWORK, * )
```

```
..
```

Purpose:

=====

ZLARFB applies a complex block reflector H or its transpose H**H to a complex M-by-N matrix C, from either the left or the right.

Arguments:

=====

[in] SIDE

SIDE is CHARACTER*1

= 'L': apply H or H**H from the Left

= 'R': apply H or H**H from the Right

[in] TRANS

TRANS is CHARACTER*1
 = 'N': apply H (No transpose)
 = 'C': apply H**H (Conjugate transpose)

[in] DIRECT

DIRECT is CHARACTER*1
 Indicates how H is formed from a product of elementary reflectors
 = 'F': $H = H(1) H(2) \dots H(k)$ (Forward)
 = 'B': $H = H(k) \dots H(2) H(1)$ (Backward)

[in] STOREV

STOREV is CHARACTER*1
 Indicates how the vectors which define the elementary reflectors are stored:
 = 'C': Columnwise
 = 'R': Rowwise

[in] M

M is INTEGER
 The number of rows of the matrix C.

[in] N

N is INTEGER
 The number of columns of the matrix C.

[in] K

K is INTEGER
 The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

[in] V

V is COMPLEX*16 array, dimension
 (LDV,K) if STOREV = 'C'
 (LDV,M) if STOREV = 'R' and SIDE = 'L'
 (LDV,N) if STOREV = 'R' and SIDE = 'R'
 See Further Details.

[in] LDV

LDV is INTEGER
The leading dimension of the array V.
If STOREV = 'C' and SIDE = 'L', LDV \geq max(1,M);
if STOREV = 'C' and SIDE = 'R', LDV \geq max(1,N);
if STOREV = 'R', LDV \geq K.

[in] T

T is COMPLEX*16 array, dimension (LDT,K)
The triangular K-by-K matrix T in the representation of the
block reflector.

[in] LDT

LDT is INTEGER
The leading dimension of the array T. LDT \geq K.

[in,out] C

C is COMPLEX*16 array, dimension (LDC,N)
On entry, the M-by-N matrix C.
On exit, C is overwritten by H*C or H**H*C or C*H or C**H.

[in] LDC

LDC is INTEGER
The leading dimension of the array C. LDC \geq max(1,M).

[out] WORK

WORK is COMPLEX*16 array, dimension (LDWORK,K)

[in] LDWORK

LDWORK is INTEGER
The leading dimension of the array WORK.
If SIDE = 'L', LDWORK \geq max(1,N);
if SIDE = 'R', LDWORK \geq max(1,M).

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Further Details:

=====

The shape of the matrix V and the storage of the vectors which define the $H(i)$ is best illustrated by the following example with $n = 5$ and $k = 3$. The elements equal to 1 are not stored; the corresponding array elements are modified but restored on exit. The rest of the array is not used.

DIRECT = 'F' and STOREV = 'C':

$$V = \begin{pmatrix} 1 & & \\ v1 & 1 & \\ v1 & v2 & 1 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \end{pmatrix}$$

DIRECT = 'F' and STOREV = 'R':

$$V = \begin{pmatrix} 1 & v1 & v1 & v1 & v1 \\ & 1 & v2 & v2 & v2 \\ & & 1 & v3 & v3 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'C':

$$V = \begin{pmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ 1 & v2 & v3 \\ & 1 & v3 \\ & & 1 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'R':

$$V = \begin{pmatrix} v1 & v1 & 1 & & \\ v2 & v2 & v2 & 1 & \\ v3 & v3 & v3 & v3 & 1 \end{pmatrix}$$

— zlarfb.f —

```
* =====
*      SUBROUTINE ZLARFB( SIDE, TRANS, DIRECT, STOREV, M, N, K, V, LDV,
*      $                  T, LDT, C, LDC, WORK, LDWORK )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          DIRECT, SIDE, STOREV, TRANS
*      INTEGER            K, LDC, LDT, LDV, LDWORK, M, N
*      ..
*      .. Array Arguments ..
*      COMPLEX*16         C( LDC, * ), T( LDT, * ), V( LDV, * ),
*      $                  WORK( LDWORK, * )
*      ..
*
*      =====
*
```

```

*      .. Parameters ..
      COMPLEX*16      ONE
      PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Local Scalars ..
      CHARACTER      TRANST
      INTEGER        I, J, LASTV, LASTC
*
*      ..
*      .. External Functions ..
      LOGICAL        LSAME
      INTEGER        ILAZLR, ILAZLC
      EXTERNAL       LSAME, ILAZLR, ILAZLC
*
*      ..
*      .. External Subroutines ..
      EXTERNAL       ZCOPY, ZGEMM, ZLACGV, ZTRMM
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC      DCONJG
*
*      ..
*      .. Executable Statements ..
*
      Quick return if possible
*
      IF( M.LE.0 .OR. N.LE.0 )
$      RETURN
*
      IF( LSAME( TRANS, 'N' ) ) THEN
        TRANST = 'C'
      ELSE
        TRANST = 'N'
      END IF
*
      IF( LSAME( STOREV, 'C' ) ) THEN
*
        IF( LSAME( DIRECT, 'F' ) ) THEN
*
          Let V = ( V1 )      (first K rows)
                  ( V2 )
          where V1 is unit lower triangular.
*
          IF( LSAME( SIDE, 'L' ) ) THEN
*
            Form H * C or H**H * C where C = ( C1 )
                                                ( C2 )
*
            LASTV = MAX( K, ILAZLR( M, K, V, LDV ) )
            LASTC = ILAZLC( LASTV, N, C, LDC )
*
            W := C**H * V = (C1**H * V1 + C2**H * V2) (stored in WORK)
*

```

```

*          W := C1**H
*
DO 10 J = 1, K
    CALL ZCOPY( LASTC, C( J, 1 ), LDC, WORK( 1, J ), 1 )
    CALL ZLACGV( LASTC, WORK( 1, J ), 1 )
10    CONTINUE
*
*          W := W * V1
*
*          CALL ZTRMM( 'Right', 'Lower', 'No transpose', 'Unit',
$              LASTC, K, ONE, V, LDV, WORK, LDWORK )
IF( LASTV.GT.K ) THEN
*
*          W := W + C2**H *V2
*
*          CALL ZGEMM( 'Conjugate transpose', 'No transpose',
$              LASTC, K, LASTV-K, ONE, C( K+1, 1 ), LDC,
$              V( K+1, 1 ), LDV, ONE, WORK, LDWORK )
END IF
*
*          W := W * T**H or W * T
*
*          CALL ZTRMM( 'Right', 'Upper', TRANST, 'Non-unit',
$              LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*          C := C - V * W**H
*
*          IF( M.GT.K ) THEN
*
*              C2 := C2 - V2 * W**H
*
*              CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$                  LASTV-K, LASTC, K,
$                  -ONE, V( K+1, 1 ), LDV, WORK, LDWORK,
$                  ONE, C( K+1, 1 ), LDC )
END IF
*
*          W := W * V1**H
*
*          CALL ZTRMM( 'Right', 'Lower', 'Conjugate transpose',
$              'Unit', LASTC, K, ONE, V, LDV, WORK, LDWORK )
*
*          C1 := C1 - W**H
*
DO 30 J = 1, K
    DO 20 I = 1, LASTC
        C( J, I ) = C( J, I ) - DCONJG( WORK( I, J ) )
20    CONTINUE
30    CONTINUE
*

```



```

ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*      Form  $C * H$  or  $C * H^{**}H$  where  $C = ( C1 \ C2 )$ 
*
      LASTV = MAX( K, ILAZLR( N, K, V, LDV ) )
      LASTC = ILAZLR( M, LASTV, C, LDC )
*
      W := C * V = (C1*V1 + C2*V2) (stored in WORK)
*
*      W := C1
*
      DO 40 J = 1, K
          CALL ZCOPY( LASTC, C( 1, J ), 1, WORK( 1, J ), 1 )
40      CONTINUE
*
*      W := W * V1
*
      CALL ZTRMM( 'Right', 'Lower', 'No transpose', 'Unit',
$          LASTC, K, ONE, V, LDV, WORK, LDWORK )
      IF( LASTV.GT.K ) THEN
*
*          W := W + C2 * V2
*
          CALL ZGEMM( 'No transpose', 'No transpose',
$              LASTC, K, LASTV-K,
$              ONE, C( 1, K+1 ), LDC, V( K+1, 1 ), LDV,
$              ONE, WORK, LDWORK )
      END IF
*
*      W := W * T or W * T**H
*
      CALL ZTRMM( 'Right', 'Upper', TRANS, 'Non-unit',
$          LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
      C := C - W * V**H
*
      IF( LASTV.GT.K ) THEN
*
*          C2 := C2 - W * V2**H
*
          CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$              LASTC, LASTV-K, K,
$              -ONE, WORK, LDWORK, V( K+1, 1 ), LDV,
$              ONE, C( 1, K+1 ), LDC )
      END IF
*
*      W := W * V1**H
*
      CALL ZTRMM( 'Right', 'Lower', 'Conjugate transpose',
$          'Unit', LASTC, K, ONE, V, LDV, WORK, LDWORK )

```

```

*
*      C1 := C1 - W
*
*      DO 60 J = 1, K
*          DO 50 I = 1, LASTC
*              C( I, J ) = C( I, J ) - WORK( I, J )
50          CONTINUE
60      CONTINUE
      END IF
*
      ELSE
*
*      Let V = ( V1 )
*              ( V2 )      (last K rows)
*      where V2 is unit upper triangular.
*
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*          Form  $H * C$  or  $H**H * C$  where  $C = \begin{pmatrix} C1 \\ C2 \end{pmatrix}$ 
*
*          LASTV = MAX( K, ILAZLR( M, K, V, LDV ) )
*          LASTC = ILAZLC( LASTV, N, C, LDC )
*
*          W := C**H * V = (C1**H * V1 + C2**H * V2) (stored in WORK)
*
*          W := C2**H
*
*          DO 70 J = 1, K
*              CALL ZCOPY( LASTC, C( LASTV-K+J, 1 ), LDC,
$              WORK( 1, J ), 1 )
$              CALL ZLACGV( LASTC, WORK( 1, J ), 1 )
70          CONTINUE
*
*          W := W * V2
*
*          CALL ZTRMM( 'Right', 'Upper', 'No transpose', 'Unit',
$          LASTC, K, ONE, V( LASTV-K+1, 1 ), LDV,
$          WORK, LDWORK )
$          IF( LASTV.GT.K ) THEN
*
*              W := W + C1**H*V1
*
*              CALL ZGEMM( 'Conjugate transpose', 'No transpose',
$              LASTC, K, LASTV-K,
$              ONE, C, LDC, V, LDV,
$              ONE, WORK, LDWORK )
*
*          END IF
*
*          W := W * T**H or W * T

```

```

*
*      CALL ZTRMM( 'Right', 'Lower', TRANST, 'Non-unit',
$          LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*      C := C - V * W**H
*
*      IF( LASTV.GT.K ) THEN
*
*          C1 := C1 - V1 * W**H
*
*          CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$              LASTV-K, LASTC, K,
$              -ONE, V, LDV, WORK, LDWORK,
$              ONE, C, LDC )
*          END IF
*
*      W := W * V2**H
*
*      CALL ZTRMM( 'Right', 'Upper', 'Conjugate transpose',
$          'Unit', LASTC, K, ONE, V( LASTV-K+1, 1 ), LDV,
$          WORK, LDWORK )
*
*      C2 := C2 - W**H
*
*      DO 90 J = 1, K
*          DO 80 I = 1, LASTC
*              C( LASTV-K+J, I ) = C( LASTV-K+J, I ) -
$                  DCONJG( WORK( I, J ) )
*          80 CONTINUE
*      90 CONTINUE
*
*      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*          Form C * H or C * H**H where C = ( C1 C2 )
*
*          LASTV = MAX( K, ILAZLR( N, K, V, LDV ) )
*          LASTC = ILAZLR( M, LASTV, C, LDC )
*
*          W := C * V = (C1*V1 + C2*V2) (stored in WORK)
*
*          W := C2
*
*          DO 100 J = 1, K
*              CALL ZCOPY( LASTC, C( 1, LASTV-K+J ), 1,
$                  WORK( 1, J ), 1 )
*          100 CONTINUE
*
*          W := W * V2
*
*          CALL ZTRMM( 'Right', 'Upper', 'No transpose', 'Unit',

```

```

$          LASTC, K, ONE, V( LASTV-K+1, 1 ), LDV,
$          WORK, LDWORK )
IF( LASTV.GT.K ) THEN
*
*          W := W + C1 * V1
*
$          CALL ZGEMM( 'No transpose', 'No transpose',
$          LASTC, K, LASTV-K,
$          ONE, C, LDC, V, LDV, ONE, WORK, LDWORK )
END IF
*
*          W := W * T or W * T**H
*
$          CALL ZTRMM( 'Right', 'Lower', TRANS, 'Non-unit',
$          LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*          C := C - W * V**H
*
IF( LASTV.GT.K ) THEN
*
*          C1 := C1 - W * V1**H
*
$          CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$          LASTC, LASTV-K, K, -ONE, WORK, LDWORK, V, LDV,
$          ONE, C, LDC )
END IF
*
*          W := W * V2**H
*
$          CALL ZTRMM( 'Right', 'Upper', 'Conjugate transpose',
$          'Unit', LASTC, K, ONE, V( LASTV-K+1, 1 ), LDV,
$          WORK, LDWORK )
*
*          C2 := C2 - W
*
DO 120 J = 1, K
DO 110 I = 1, LASTC
C( I, LASTV-K+J ) = C( I, LASTV-K+J )
$          - WORK( I, J )
110 CONTINUE
120 CONTINUE
END IF
END IF
*
ELSE IF( LSAME( STOREV, 'R' ) ) THEN
*
IF( LSAME( DIRECT, 'F' ) ) THEN
*
*          Let V = ( V1 V2 ) (V1: first K columns)
*          where V1 is unit upper triangular.

```

```

*
      IF( LSAME( SIDE, 'L' ) ) THEN
*
*         Form  $H * C$  or  $H^{**H} * C$  where  $C = \begin{pmatrix} C1 \\ C2 \end{pmatrix}$ 
*
*         LASTV = MAX( K, ILAZLC( K, M, V, LDV ) )
*         LASTC = ILAZLC( LASTV, N, C, LDC )
*
*         W := C**H * V**H = (C1**H * V1**H + C2**H * V2**H) (stored in WORK)
*
*         W := C1**H
*
      DO 130 J = 1, K
          CALL ZCOPY( LASTC, C( J, 1 ), LDC, WORK( 1, J ), 1 )
          CALL ZLACGV( LASTC, WORK( 1, J ), 1 )
130      CONTINUE
*
*         W := W * V1**H
*
*         CALL ZTRMM( 'Right', 'Upper', 'Conjugate transpose',
$           'Unit', LASTC, K, ONE, V, LDV, WORK, LDWORK )
*         IF( LASTV.GT.K ) THEN
*
*             W := W + C2**H * V2**H
*
*             CALL ZGEMM( 'Conjugate transpose',
$               'Conjugate transpose', LASTC, K, LASTV-K,
$               ONE, C( K+1, 1 ), LDC, V( 1, K+1 ), LDV,
$               ONE, WORK, LDWORK )
*             END IF
*
*         W := W * T**H or W * T
*
*         CALL ZTRMM( 'Right', 'Upper', TRANST, 'Non-unit',
$           LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*         C := C - V**H * W**H
*
*         IF( LASTV.GT.K ) THEN
*
*             C2 := C2 - V2**H * W**H
*
*             CALL ZGEMM( 'Conjugate transpose',
$               'Conjugate transpose', LASTV-K, LASTC, K,
$               -ONE, V( 1, K+1 ), LDV, WORK, LDWORK,
$               ONE, C( K+1, 1 ), LDC )
*             END IF
*
*         W := W * V1

```

```

*
*      CALL ZTRMM( 'Right', 'Upper', 'No transpose', 'Unit',
$          LASTC, K, ONE, V, LDV, WORK, LDWORK )
*
*      C1 := C1 - W**H
*
*      DO 150 J = 1, K
*          DO 140 I = 1, LASTC
*              C( J, I ) = C( J, I ) - DCONJG( WORK( I, J ) )
140          CONTINUE
150      CONTINUE
*
*      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*          Form  $C * H$  or  $C * H^{**H}$  where  $C = ( C1 \ C2 )$ 
*
*          LASTV = MAX( K, ILAZLC( K, N, V, LDV ) )
*          LASTC = ILAZLR( M, LASTV, C, LDC )
*
*          W := C * V**H = (C1*V1**H + C2*V2**H) (stored in WORK)
*
*          W := C1
*
*          DO 160 J = 1, K
*              CALL ZCOPY( LASTC, C( 1, J ), 1, WORK( 1, J ), 1 )
160          CONTINUE
*
*          W := W * V1**H
*
*          CALL ZTRMM( 'Right', 'Upper', 'Conjugate transpose',
$              'Unit', LASTC, K, ONE, V, LDV, WORK, LDWORK )
*          IF( LASTV.GT.K ) THEN
*
*              W := W + C2 * V2**H
*
*              CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$                  LASTC, K, LASTV-K, ONE, C( 1, K+1 ), LDC,
$                  V( 1, K+1 ), LDV, ONE, WORK, LDWORK )
*          END IF
*
*          W := W * T or W * T**H
*
*          CALL ZTRMM( 'Right', 'Upper', TRANS, 'Non-unit',
$              LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*          C := C - W * V
*
*          IF( LASTV.GT.K ) THEN
*
*              C2 := C2 - W * V2

```

```

*
*      CALL ZGEMM( 'No transpose', 'No transpose',
$          LASTC, LASTV-K, K,
$          -ONE, WORK, LDWORK, V( 1, K+1 ), LDV,
$          ONE, C( 1, K+1 ), LDC )
*      END IF
*
*      W := W * V1
*
*      CALL ZTRMM( 'Right', 'Upper', 'No transpose', 'Unit',
$          LASTC, K, ONE, V, LDV, WORK, LDWORK )
*
*      C1 := C1 - W
*
*      DO 180 J = 1, K
*          DO 170 I = 1, LASTC
*              C( I, J ) = C( I, J ) - WORK( I, J )
170          CONTINUE
180      CONTINUE
*
*      END IF
*
*      ELSE
*
*      Let V = ( V1 V2 )    (V2: last K columns)
*      where V2 is unit lower triangular.
*
*      IF( LSAME( SIDE, 'L' ) ) THEN
*
*          Form H * C or H**H * C where C = ( C1 )
*                                           ( C2 )
*
*          LASTV = MAX( K, ILAZLC( K, M, V, LDV ) )
*          LASTC = ILAZLC( LASTV, N, C, LDC )
*
*          W := C**H * V**H = (C1**H * V1**H + C2**H * V2**H) (stored in WORK)
*
*          W := C2**H
*
*          DO 190 J = 1, K
*              CALL ZCOPY( LASTC, C( LASTV-K+J, 1 ), LDC,
$                  WORK( 1, J ), 1 )
*              CALL ZLACGV( LASTC, WORK( 1, J ), 1 )
190          CONTINUE
*
*          W := W * V2**H
*
*      CALL ZTRMM( 'Right', 'Lower', 'Conjugate transpose',
$          'Unit', LASTC, K, ONE, V( 1, LASTV-K+1 ), LDV,
$          WORK, LDWORK )

```

```

IF( LASTV.GT.K ) THEN
*
*      W := W + C1**H * V1**H
*
      CALL ZGEMM( 'Conjugate transpose',
$          'Conjugate transpose', LASTC, K, LASTV-K,
$          ONE, C, LDC, V, LDV, ONE, WORK, LDWORK )
      END IF
*
*      W := W * T**H or W * T
*
      CALL ZTRMM( 'Right', 'Lower', TRANST, 'Non-unit',
$          LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*      C := C - V**H * W**H
*
      IF( LASTV.GT.K ) THEN
*
*          C1 := C1 - V1**H * W**H
*
          CALL ZGEMM( 'Conjugate transpose',
$              'Conjugate transpose', LASTV-K, LASTC, K,
$              -ONE, V, LDV, WORK, LDWORK, ONE, C, LDC )
          END IF
*
*      W := W * V2
*
      CALL ZTRMM( 'Right', 'Lower', 'No transpose', 'Unit',
$          LASTC, K, ONE, V( 1, LASTV-K+1 ), LDV,
$          WORK, LDWORK )
*
*      C2 := C2 - W**H
*
      DO 210 J = 1, K
          DO 200 I = 1, LASTC
              C( LASTV-K+J, I ) = C( LASTV-K+J, I ) -
$                  DCONJG( WORK( I, J ) )
          200      CONTINUE
          210      CONTINUE
*
      ELSE IF( LSAME( SIDE, 'R' ) ) THEN
*
*          Form C * H or C * H**H where C = ( C1 C2 )
*
          LASTV = MAX( K, ILAZLC( K, N, V, LDV ) )
          LASTC = ILAZLR( M, LASTV, C, LDC )
*
*          W := C * V**H = (C1*V1**H + C2*V2**H) (stored in WORK)
*
*          W := C2

```



```

*
      DO 220 J = 1, K
          CALL ZCOPY( LASTC, C( 1, LASTV-K+J ), 1,
$              WORK( 1, J ), 1 )
220      CONTINUE
*
*      W := W * V2**H
*
      CALL ZTRMM( 'Right', 'Lower', 'Conjugate transpose',
$          'Unit', LASTC, K, ONE, V( 1, LASTV-K+1 ), LDV,
$          WORK, LDWORK )
      IF( LASTV.GT.K ) THEN
*
*      W := W + C1 * V1**H
*
      CALL ZGEMM( 'No transpose', 'Conjugate transpose',
$          LASTC, K, LASTV-K, ONE, C, LDC, V, LDV, ONE,
$          WORK, LDWORK )
      END IF
*
*      W := W * T or W * T**H
*
      CALL ZTRMM( 'Right', 'Lower', TRANS, 'Non-unit',
$          LASTC, K, ONE, T, LDT, WORK, LDWORK )
*
*      C := C - W * V
*
      IF( LASTV.GT.K ) THEN
*
*      C1 := C1 - W * V1
*
      CALL ZGEMM( 'No transpose', 'No transpose',
$          LASTC, LASTV-K, K, -ONE, WORK, LDWORK, V, LDV,
$          ONE, C, LDC )
      END IF
*
*      W := W * V2
*
      CALL ZTRMM( 'Right', 'Lower', 'No transpose', 'Unit',
$          LASTC, K, ONE, V( 1, LASTV-K+1 ), LDV,
$          WORK, LDWORK )
*
*      C1 := C1 - W
*
      DO 240 J = 1, K
          DO 230 I = 1, LASTC
              C( I, LASTV-K+J ) = C( I, LASTV-K+J )
$                  - WORK( I, J )
230      CONTINUE
240      CONTINUE

```

```

*
*           END IF
*
*           END IF
*       END IF
*
*       RETURN
*
*       End of ZLARFB
*
*       END

```

— LAPACK zlarfb —

```

(let* ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (ignorable one))
  (defun zlarfb (side trans direct storev m n k v ldv t$ ldt c ldc work ldwork)
    (declare (type (simple-array character (*)) storev direct trans side)
      (type (f2cl-lib:integer4) ldwork ldc ldt ldv k n m)
      (type (array f2cl-lib:complex16 (*)) work c t$ v))
    (f2cl-lib:with-multi-array-data
      ((v f2cl-lib:complex16 v-%data% v-%offset%)
       (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
       (c f2cl-lib:complex16 c-%data% c-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%)
       (side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%)
       (direct character direct-%data% direct-%offset%)
       (storev character storev-%data% storev-%offset%))
      (prog
        ((i 0) (j 0) (lastv 0) (lastc 0)
         (transt (make-array '(1) :element-type 'character
                             :initial-element #\space)))
        (declare (type (f2cl-lib:integer4) lastc lastv j i)
          (type (simple-array character (1)) transt))
        (if (or (<= m 0) (<= n 0)) (go end_label))
        (cond
          ((multiple-value-bind (ret-val var-0 var-1) (lsame trans "N")
            (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val)
           (f2cl-lib:f2cl-set-string transt "C" (string 1)))
          (t (f2cl-lib:f2cl-set-string transt "N" (string 1))))
        (cond
          ((multiple-value-bind (ret-val var-0 var-1) (lsame storev "C")
            (declare (ignore var-1)) (when var-0 (setf storev var-0)) ret-val)
           (cond
            ((multiple-value-bind (ret-val var-0 var-1) (lsame direct "F")

```

```

(declare (ignore var-1)) (when var-0 (setf direct var-0)) ret-val)
(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
    (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
    (setf lastv
      (max (the f2cl-lib:integer4 k)
        (the f2cl-lib:integer4 (ilazlr m k v ldv))))
    (setf lastc (ilazlc lastv n c ldc))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j k) nil)
      (tagbody
        (multiple-value-bind (var-0 var-1 var-2
                              var-3 var-4)
          (zcopy lastc
            (f2cl-lib:array-slice c-%data%
              f2cl-lib:complex16 (j 1)
              ((1 ldc) (1 *)) c-%offset%)
            ldc
            (f2cl-lib:array-slice work-%data%
              f2cl-lib:complex16 (1 j)
              ((1 ldwork) (1 *)) work-%offset%)
            1)
          (declare (ignore var-1 var-3 var-4))
          (when var-0 (setf lastc var-0))
          (when var-2 (setf ldc var-2)))
        (zlacgv lastc
          (f2cl-lib:array-slice work-%data%
            f2cl-lib:complex16 (1 j)
            ((1 ldwork) (1 *)) work-%offset%)
          1)
        label10))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
      (ztrmm "Right" "Lower" "No transpose" "Unit"
        lastc k one v ldv work
        ldwork)
      (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
      (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
      (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
      (when var-10 (setf ldwork var-10)))
    (cond
      ((> lastv k)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6
           var-7 var-8 var-9 var-10
           var-11 var-12)
          (zgemm "Conjugate transpose" "No transpose" lastc k
            (f2cl-lib:int-sub lastv k) one
            (f2cl-lib:array-slice c-%data% f2cl-lib:complex16 ((+ k 1) 1)

```

```

      ((1 ldc) (1 *)) c-%offset%)
    ldc
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16 ((+ k 1) 1)
      ((1 ldv) (1 *)) v-%offset%)
    ldv one work ldwork)
  (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
  (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
  (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
  (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
  (when var-12 (setf ldwork var-12))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Upper" transt "Non-unit" lastc k one t$ ldt work
    ldwork)
  (declare (ignore var-0 var-1 var-3 var-7 var-9))
  (when var-2 (setf transt var-2)) (when var-4 (setf lastc var-4))
  (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
  (when var-8 (setf ldt var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  (> m k)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10
     var-11 var-12)
    (zgemm "No transpose" "Conjugate transpose"
      (f2cl-lib:int-sub lastv k) lastc k (- one)
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16 ((+ k 1) 1)
        ((1 ldv) (1 *)) v-%offset%)
      ldv work ldwork one
      (f2cl-lib:array-slice c-%data% f2cl-lib:complex16 ((+ k 1) 1)
        ((1 ldc) (1 *)) c-%offset%)
      ldc)
    (declare (ignore var-0 var-1 var-2 var-5 var-6 var-8 var-11))
    (when var-3 (setf lastc var-3)) (when var-4 (setf k var-4))
    (when var-7 (setf ldv var-7)) (when var-9 (setf ldwork var-9))
    (when var-10 (setf one var-10))
    (when var-12 (setf ldc var-12))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Lower" "Conjugate transpose" "Unit"
    lastc k one v ldv
    work ldwork)
  (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
  (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
  (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
  (when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))

```

```

(> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    (> i lastc) nil)
  (tagbody
    (setf (f2cl-lib:fref c-%data% (j i)
      ((1 ldc) (1 *)) c-%offset%)
      (coerce
        (- (f2cl-lib:fref c-%data% (j i)
          ((1 ldc) (1 *)) c-%offset%)
          (f2cl-lib:dconjg
            (f2cl-lib:fref work-%data% (i j)
              ((1 ldwork) (1 *))
              work-%offset%)))
          'f2cl-lib:complex16))
      label20))
    label30)))
((multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
  (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
  (setf lastv
    (max (the f2cl-lib:integer4 k)
      (the f2cl-lib:integer4 (ilazlr n k v ldv)))))
  (setf lastc (ilazlr m lastv c ldc))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j k) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2
      var-3 var-4)
      (zcopy lastc
        (f2cl-lib:array-slice c-%data%
          f2cl-lib:complex16 (1 j)
          ((1 ldc) (1 *)) c-%offset%)
        1
        (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 (1 j)
          ((1 ldwork) (1 *)) work-%offset%)
        1)
      (declare (ignore var-1 var-2 var-3 var-4))
      (when var-0 (setf lastc var-0)))
    label40))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
  (ztrmm "Right" "Lower" "No transpose" "Unit"
    lastc k one v ldv work
    ldwork)
  (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
  (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
  (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
  (when var-10 (setf ldwork var-10)))

```

```

(cond
  (> lastv k)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10
     var-11 var-12)
    (zgemm "No transpose" "No transpose" lastc k
      (f2cl-lib:int-sub lastv k) one
      (f2cl-lib:array-slice c-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldc) (1 *)) c-%offset%)
      ldc
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16 ((+ k 1) 1)
        ((1 ldv) (1 *)) v-%offset%)
      ldv one work ldwork)
    (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
    (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
    (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
    (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
    (when var-12 (setf ldwork var-12))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Upper" trans "Non-unit"
    lastc k one t$ ldt work ldwork)
  (declare (ignore var-0 var-1 var-3 var-7 var-9))
  (when var-2 (setf trans var-2)) (when var-4 (setf lastc var-4))
  (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
  (when var-8 (setf ldt var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  (> lastv k)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10
     var-11 var-12)
    (zgemm "No transpose" "Conjugate transpose" lastc
      (f2cl-lib:int-sub lastv k) k (- one) work ldwork
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16 ((+ k 1) 1)
        ((1 ldv) (1 *)) v-%offset%)
      ldv one
      (f2cl-lib:array-slice c-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldc) (1 *)) c-%offset%)
      ldc)
    (declare (ignore var-0 var-1 var-3 var-5 var-6 var-8 var-11))
    (when var-2 (setf lastc var-2)) (when var-4 (setf k var-4))
    (when var-7 (setf ldwork var-7)) (when var-9 (setf ldv var-9))
    (when var-10 (setf one var-10))
    (when var-12 (setf ldc var-12))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7

```

```

var-8 var-9 var-10)
(ztrmm "Right" "Lower" "Conjugate transpose" "Unit"
 lastc k one v ldv
 work ldwork)
(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
(when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
(when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i lastc) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data% (i j)
          ((1 ldc) (1 *)) c-%offset%)
          (- (f2cl-lib:fref c-%data% (i j)
            ((1 ldc) (1 *)) c-%offset%)
            (f2cl-lib:fref work-%data% (i j)
              ((1 ldwork) (1 *)) work-%offset%)))
          label50))
        label60))))))
(t
  (cond
    ((multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
      (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
      (setf lastv
        (max (the f2cl-lib:integer4 k)
          (the f2cl-lib:integer4 (ilazlr m k v ldv))))
      (setf lastc (ilazlc lastv n c ldc))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j k) nil)
        (tagbody
          (multiple-value-bind (var-0 var-1 var-2
            var-3 var-4)
            (zcopy lastc
              (f2cl-lib:array-slice c-%data%
                f2cl-lib:complex16
                ((+ lastv (f2cl-lib:int-sub k) j) 1)
                ((1 ldc) (1 *)) c-%offset%)
              ldc
              (f2cl-lib:array-slice work-%data%
                f2cl-lib:complex16 (1 j)
                ((1 ldwork) (1 *)) work-%offset%)
              1)
            (declare (ignore var-1 var-3 var-4))
            (when var-0 (setf lastc var-0))
            (when var-2 (setf ldc var-2)))
          (zlagv lastc
            (f2cl-lib:array-slice work-%data%
              (f2cl-lib:array-slice c-%data%
                f2cl-lib:complex16
                ((+ lastv (f2cl-lib:int-sub k) j) 1)
                ((1 ldc) (1 *)) c-%offset%)
              ldc
              (f2cl-lib:array-slice work-%data%
                f2cl-lib:complex16 (1 j)
                ((1 ldwork) (1 *)) work-%offset%)
              1)
            1)
          label60))))))
  )

```

```

        f2cl-lib:complex16 (1 j)
        ((1 ldwork) (1 *)) work-%offset%)
    1)
    label70))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (ztrmm "Right" "Upper" "No transpose" "Unit" lastc k one
  (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
   ((+ lastv (f2cl-lib:int-sub k) 1) 1)
   ((1 ldv) (1 *)) v-%offset%)
  ldv work ldwork)
 (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
 (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
 (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
 (when var-10 (setf ldwork var-10)))
(cond
 (> lastv k)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10
   var-11 var-12)
  (zgemm "Conjugate transpose" "No transpose" lastc k
   (f2cl-lib:int-sub lastv k) one c ldc v ldv one work ldwork)
  (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
  (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
  (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
  (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
  (when var-12 (setf ldwork var-12))))))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (ztrmm "Right" "Lower" transt "Non-unit" lastc k one t$ ldt work
  ldwork)
 (declare (ignore var-0 var-1 var-3 var-7 var-9))
 (when var-2 (setf transt var-2)) (when var-4 (setf lastc var-4))
 (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
 (when var-8 (setf ldt var-8))
 (when var-10 (setf ldwork var-10)))
(cond
 (> lastv k)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10
   var-11 var-12)
  (zgemm "No transpose" "Conjugate transpose"
   (f2cl-lib:int-sub lastv k) lastc k
   (- one) v ldv work ldwork one c
   ldc)
  (declare (ignore var-0 var-1 var-2 var-5 var-6 var-8 var-11))

```



```

      (when var-3 (setf lastc var-3)) (when var-4 (setf k var-4))
      (when var-7 (setf ldv var-7)) (when var-9 (setf ldwork var-9))
      (when var-10 (setf one var-10))
      (when var-12 (setf ldc var-12))))))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (ztrmm "Right" "Upper" "Conjugate transpose" "Unit" lastc k one
  (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
   ((+ lastv (f2cl-lib:int-sub k) 1) 1)
   ((1 ldv) (1 *)) v-%offset%)
  ldv work ldwork)
 (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
 (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
 (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
 (when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i lastc) nil)
    (tagbody
      (setf
        (f2cl-lib:fref c-%data%
          ((f2cl-lib:int-add
            (f2cl-lib:int-sub lastv k) j) i)
          ((1 ldc) (1 *))
          c-%offset%)
        (coerce
          (-
            (f2cl-lib:fref c-%data%
              ((f2cl-lib:int-add
                (f2cl-lib:int-sub lastv k) j) i)
              ((1 ldc) (1 *)) c-%offset%)
            (f2cl-lib:dconjg
              (f2cl-lib:fref work-%data% (i j)
                ((1 ldwork) (1 *))
                work-%offset%)))
            'f2cl-lib:complex16))
          label80))
      label90)))
((multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
 (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
 (setf lastv
  (max (the f2cl-lib:integer4 k)
   (the f2cl-lib:integer4 (ilazlr n k v ldv)))))
 (setf lastc (ilazlr m lastv c ldc))
 (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
 (> j k) nil)
  (tagbody

```

```

(multiple-value-bind (var-0 var-1 var-2
                     var-3 var-4)
  (zcopy lastc
    (f2cl-lib:array-slice c-%data%
      f2cl-lib:complex16
      (1 (f2cl-lib:int-add
        (f2cl-lib:int-sub lastv k) j))
        ((1 ldc) (1 *)))
      c-%offset%)
    1
    (f2cl-lib:array-slice work-%data%
      f2cl-lib:complex16 (1 j)
      ((1 ldwork) (1 *)) work-%offset%)
    1)
  (declare (ignore var-1 var-2 var-3 var-4))
  (when var-0 (setf lastc var-0)))
label100))

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Upper" "No transpose" "Unit" lastc k one
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
      ((+ lastv (f2cl-lib:int-sub k) 1) 1)
      ((1 ldv) (1 *)) v-%offset%)
    ldv work ldwork)
  (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
  (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
  (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  ((> lastv k)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10
       var-11 var-12)
      (zgemm "No transpose" "No transpose" lastc k
        (f2cl-lib:int-sub lastv k) one c ldc v ldv one work ldwork)
      (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
      (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
      (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
      (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
      (when var-12 (setf ldwork var-12))))))

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Lower" trans "Non-unit"
    lastc k one t$ ldt work ldwork)
  (declare (ignore var-0 var-1 var-3 var-7 var-9))
  (when var-2 (setf trans var-2)) (when var-4 (setf lastc var-4))
  (when var-5 (setf k var-5)) (when var-6 (setf one var-6))

```

```

      (when var-8 (setf ldt var-8))
      (when var-10 (setf ldwork var-10)))
    (cond
      ((> lastv k)
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
           var-9 var-10
           var-11 var-12)
          (zgemm "No transpose" "Conjugate transpose" lastc
            (f2cl-lib:int-sub lastv k) k (- one) work
            ldwork v ldv one c ldc)
          (declare (ignore var-0 var-1 var-3 var-5 var-6 var-8 var-11))
          (when var-2 (setf lastc var-2)) (when var-4 (setf k var-4))
          (when var-7 (setf ldwork var-7)) (when var-9 (setf ldv var-9))
          (when var-10 (setf one var-10))
          (when var-12 (setf ldc var-12))))))
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
         var-8 var-9 var-10)
        (ztrmm "Right" "Upper" "Conjugate transpose" "Unit" lastc k one
          (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
            ((+ lastv (f2cl-lib:int-sub k) 1) 1)
            ((1 ldv) (1 *)) v-%offset%)
          ldv work ldwork)
        (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
        (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
        (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
        (when var-10 (setf ldwork var-10)))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j k) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i lastc) nil)
            (tagbody
              (setf
                (f2cl-lib:fref c-%data%
                  (i (f2cl-lib:int-add
                    (f2cl-lib:int-sub lastv k) j))
                    ((1 ldc) (1 *))
                    c-%offset%)
                (-
                  (f2cl-lib:fref c-%data%
                    (i (f2cl-lib:int-add
                      (f2cl-lib:int-sub lastv k) j))
                      ((1 ldc) (1 *))
                      c-%offset%)
                  (f2cl-lib:fref work-%data% (i j)
                    ((1 ldwork) (1 *)) work-%offset%)))
                label110))
              label120))))))

```

```

((multiple-value-bind (ret-val var-0 var-1) (lsame storev "R")
  (declare (ignore var-1)) (when var-0 (setf storev var-0)) ret-val)
(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame direct "F")
    (declare (ignore var-1)) (when var-0 (setf direct var-0)) ret-val)
  (cond
    ((multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
      (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
    (setf lastv
      (max (the f2cl-lib:integer4 k)
        (the f2cl-lib:integer4 (ilazlc k m v ldv))))
    (setf lastc (ilazlc lastv n c ldc))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j k) nil)
    (tagbody
      (multiple-value-bind (var-0 var-1 var-2
        var-3 var-4)
        (zcopy lastc
          (f2cl-lib:array-slice c-%data%
            f2cl-lib:complex16 (j 1)
            ((1 ldc) (1 *)) c-%offset%)
          ldc
          (f2cl-lib:array-slice work-%data%
            f2cl-lib:complex16 (1 j)
            ((1 ldwork) (1 *)) work-%offset%)
          1)
        (declare (ignore var-1 var-3 var-4))
        (when var-0 (setf lastc var-0))
        (when var-2 (setf ldc var-2)))
      (zlacgv lastc
        (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 (1 j)
          ((1 ldwork) (1 *)) work-%offset%)
        1)
      label130))
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
        var-8 var-9 var-10)
      (ztrmm "Right" "Upper" "Conjugate transpose" "Unit"
        lastc k one v ldv
        work ldwork)
      (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
      (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
      (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
      (when var-10 (setf ldwork var-10)))
    (cond
      (> lastv k)
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
          var-8 var-9 var-10)

```

```

var-11 var-12)
(zgemm "Conjugate transpose" "Conjugate transpose" lastc k
(f2cl-lib:int-sub lastv k) one
(f2cl-lib:array-slice c-%data% f2cl-lib:complex16 ((+ k 1) 1)
((1 ldc) (1 *)) c-%offset%)
ldc
(f2cl-lib:array-slice v-%data% f2cl-lib:complex16
(1 (f2cl-lib:int-add k 1)) ((1 ldv) (1 *)) v-%offset%)
ldv one work ldwork)
(declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
(when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
(when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
(when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
(when var-12 (setf ldwork var-12))))))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10)
(ztrmm "Right" "Upper" transt "Non-unit" lastc k one t$ ldc work
ldwork)
(declare (ignore var-0 var-1 var-3 var-7 var-9))
(when var-2 (setf transt var-2)) (when var-4 (setf lastc var-4))
(when var-5 (setf k var-5)) (when var-6 (setf one var-6))
(when var-8 (setf ldc var-8))
(when var-10 (setf ldwork var-10)))
(cond
(> lastv k)
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10
var-11 var-12)
(zgemm "Conjugate transpose" "Conjugate transpose"
(f2cl-lib:int-sub lastv k) lastc k (- one)
(f2cl-lib:array-slice v-%data% f2cl-lib:complex16
(1 (f2cl-lib:int-add k 1)) ((1 ldv) (1 *)) v-%offset%)
ldv work ldwork one
(f2cl-lib:array-slice c-%data% f2cl-lib:complex16 ((+ k 1) 1)
((1 ldc) (1 *)) c-%offset%)
ldc)
(declare (ignore var-0 var-1 var-2 var-5 var-6 var-8 var-11))
(when var-3 (setf lastc var-3)) (when var-4 (setf k var-4))
(when var-7 (setf ldv var-7)) (when var-9 (setf ldwork var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldc var-12))))))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
var-8 var-9 var-10)
(ztrmm "Right" "Upper" "No transpose" "Unit"
lastc k one v ldv work
ldwork)
(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))

```

```

(when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
(when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
(when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i lastc) nil)
      (tagbody
        (setf (f2cl-lib:fref c-%data% (j i)
          ((1 ldc) (1 *)) c-%offset%)
          (coerce
            (- (f2cl-lib:fref c-%data% (j i)
              ((1 ldc) (1 *)) c-%offset%)
              (f2cl-lib:dconjg
                (f2cl-lib:fref work-%data% (i j)
                  ((1 ldwork) (1 *))
                  work-%offset%)))
              'f2cl-lib:complex16))
            label140))
          label150)))
((multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
  (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
  (setf lastv
    (max (the f2cl-lib:integer4 k)
      (the f2cl-lib:integer4 (ilazlc k n v ldv))))
  (setf lastc (ilazlr m lastv c ldc))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    (> j k) nil)
    (tagbody
      (multiple-value-bind (var-0 var-1 var-2
        var-3 var-4)
        (zcopy lastc
          (f2cl-lib:array-slice c-%data%
            f2cl-lib:complex16 (1 j)
            ((1 ldc) (1 *)) c-%offset%)
            1
          (f2cl-lib:array-slice work-%data%
            f2cl-lib:complex16 (1 j)
            ((1 ldwork) (1 *)) work-%offset%)
            1)
          (declare (ignore var-1 var-2 var-3 var-4))
          (when var-0 (setf lastc var-0)))
        label160))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10)
    (ztrmm "Right" "Upper" "Conjugate transpose" "Unit"
      lastc k one v ldv
      work ldwork)

```

```

(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
(when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
(when var-10 (setf ldwork var-10)))
(cond
  ((> lastv k)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10
      var-11 var-12)
     (zgemm "No transpose" "Conjugate transpose" lastc k
      (f2cl-lib:int-sub lastv k) one
      (f2cl-lib:array-slice c-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldc) (1 *)) c-%offset%)
      ldc
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldv) (1 *)) v-%offset%)
      ldv one work ldwork)
     (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
     (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
     (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
     (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
     (when var-12 (setf ldwork var-12))))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Upper" trans "Non-unit"
   lastc k one t$ ldt work ldwork)
  (declare (ignore var-0 var-1 var-3 var-7 var-9))
  (when var-2 (setf trans var-2)) (when var-4 (setf lastc var-4))
  (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
  (when var-8 (setf ldt var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  ((> lastv k)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10
      var-11 var-12)
     (zgemm "No transpose" "No transpose"
      lastc (f2cl-lib:int-sub lastv k)
      k (- one) work ldwork
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldv) (1 *)) v-%offset%)
      ldv one
      (f2cl-lib:array-slice c-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldc) (1 *)) c-%offset%)
      ldc)
     (declare (ignore var-0 var-1 var-3 var-5 var-6 var-8 var-11))
     (when var-2 (setf lastc var-2)) (when var-4 (setf k var-4))

```

```

      (when var-7 (setf ldwork var-7)) (when var-9 (setf ldv var-9))
      (when var-10 (setf one var-10))
      (when var-12 (setf ldc var-12))))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
 var-8 var-9 var-10)
 (ztrmm "Right" "Upper" "No transpose" "Unit"
  lastc k one v ldv work
  ldwork)
 (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
 (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
 (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
 (when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i lastc) nil)
    (tagbody
      (setf (f2cl-lib:fref c-%data% (i j)
        ((1 ldc) (1 *)) c-%offset%)
        (- (f2cl-lib:fref c-%data% (i j)
          ((1 ldc) (1 *)) c-%offset%)
          (f2cl-lib:fref work-%data% (i j)
            ((1 ldwork) (1 *)) work-%offset%)))
        label170))
      label180))))))
(t
 (cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
    (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
   (setf lastv
    (max (the f2cl-lib:integer4 k)
      (the f2cl-lib:integer4 (ilazlc k m v ldv))))
    (setf lastc (ilazlc lastv n c ldc))
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      (> j k) nil)
      (tagbody
        (multiple-value-bind (var-0 var-1 var-2
          var-3 var-4)
          (zcopy lastc
            (f2cl-lib:array-slice c-%data%
              f2cl-lib:complex16
              ((+ lastv (f2cl-lib:int-sub k) j) 1)
              ((1 ldc) (1 *)) c-%offset%)
            ldc
            (f2cl-lib:array-slice work-%data%
              f2cl-lib:complex16 (1 j)
              ((1 ldwork) (1 *)) work-%offset%)
            1)

```



```

        (declare (ignore var-1 var-3 var-4))
        (when var-0 (setf lastc var-0))
        (when var-2 (setf ldc var-2))
        (zlagv lastc
         (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 (1 j)
          ((1 ldwork) (1 *)) work-%offset%)
         1)
        label190))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
  var-8 var-9 var-10)
 (ztrmm "Right" "Lower" "Conjugate transpose" "Unit" lastc k one
  (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
   (1 (f2cl-lib:int-add (f2cl-lib:int-sub lastv k) 1))
   ((1 ldv) (1 *))
   v-%offset%)
  ldv work ldwork)
 (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
 (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
 (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
 (when var-10 (setf ldwork var-10)))
(cond
 (> lastv k)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10
   var-11 var-12)
  (zgemm "Conjugate transpose" "Conjugate transpose" lastc k
   (f2cl-lib:int-sub lastv k) one c ldc v ldv one work ldwork)
  (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
  (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
  (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
  (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
  (when var-12 (setf ldwork var-12))))))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
  var-9 var-10)
 (ztrmm "Right" "Lower" transt "Non-unit" lastc k one t$ ldt work
  ldwork)
 (declare (ignore var-0 var-1 var-3 var-7 var-9))
 (when var-2 (setf transt var-2)) (when var-4 (setf lastc var-4))
 (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
 (when var-8 (setf ldt var-8))
 (when var-10 (setf ldwork var-10)))
(cond
 (> lastv k)
 (multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
   var-9 var-10)

```

```

var-11 var-12)
(zgemm "Conjugate transpose" "Conjugate transpose"
(f2cl-lib:int-sub lastv k) lastc k (- one)
v ldv work ldwork one c
ldc)
(declare (ignore var-0 var-1 var-2 var-5 var-6 var-8 var-11))
(when var-3 (setf lastc var-3)) (when var-4 (setf k var-4))
(when var-7 (setf ldv var-7)) (when var-9 (setf ldwork var-9))
(when var-10 (setf one var-10))
(when var-12 (setf ldc var-12))))))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8
var-9 var-10)
(ztrmm "Right" "Lower" "No transpose" "Unit" lastc k one
(f2cl-lib:array-slice v-%data% f2cl-lib:complex16
(1 (f2cl-lib:int-add (f2cl-lib:int-sub lastv k) 1))
((1 ldv) (1 *))
v-%offset%)
ldv work ldwork)
(declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
(when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
(when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
(when var-10 (setf ldwork var-10)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
(> j k) nil)
(tagbody
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
(> i lastc) nil)
(tagbody
(setf
(f2cl-lib:fref c-%data%
((f2cl-lib:int-add
(f2cl-lib:int-sub lastv k) j) i)
((1 ldc) (1 *))
c-%offset%)
(coerce
(-
(f2cl-lib:fref c-%data%
((f2cl-lib:int-add
(f2cl-lib:int-sub lastv k) j) i)
((1 ldc) (1 *)) c-%offset%)
(f2cl-lib:dconjg
(f2cl-lib:fref work-%data% (i j)
((1 ldwork) (1 *))
work-%offset%))))
'f2cl-lib:complex16))
label200))
label210)))
((multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
(declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)

```

```

(setf lastv
  (max (the f2cl-lib:integer4 k)
        (the f2cl-lib:integer4 (ilazlc k n v ldv))))
(setf lastc (ilazlr m lastv c ldc))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j k) nil)
  (tagbody
    (multiple-value-bind (var-0 var-1 var-2
                          var-3 var-4)
      (zcopy lastc
        (f2cl-lib:array-slice c-%data%
          f2cl-lib:complex16
          (1 (f2cl-lib:int-add
              (f2cl-lib:int-sub lastv k) j))
            ((1 ldc) (1 *))
            c-%offset%)
          1
          (f2cl-lib:array-slice work-%data%
            f2cl-lib:complex16 (1 j)
            ((1 ldwork) (1 *)) work-%offset%)
          1)
        (declare (ignore var-1 var-2 var-3 var-4))
        (when var-0 (setf lastc var-0)))
      label220))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Lower" "Conjugate transpose" "Unit" lastc k one
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
      (1 (f2cl-lib:int-add (f2cl-lib:int-sub lastv k) 1))
        ((1 ldv) (1 *))
        v-%offset%)
    ldv work ldwork)
  (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
  (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
  (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  (> lastv k)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10
     var-11 var-12)
    (zgemm "No transpose" "Conjugate transpose" lastc k
      (f2cl-lib:int-sub lastv k) one c ldc v ldv one work ldwork)
    (declare (ignore var-0 var-1 var-4 var-6 var-8 var-11))
    (when var-2 (setf lastc var-2)) (when var-3 (setf k var-3))
    (when var-5 (setf one var-5)) (when var-7 (setf ldc var-7))
    (when var-9 (setf ldv var-9)) (when var-10 (setf one var-10))
    (when var-12 (setf ldwork var-12))))))

```

```

(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10)
  (ztrmm "Right" "Lower" trans "Non-unit"
    lastc k one t$ ldt work ldwork)
  (declare (ignore var-0 var-1 var-3 var-7 var-9))
  (when var-2 (setf trans var-2)) (when var-4 (setf lastc var-4))
  (when var-5 (setf k var-5)) (when var-6 (setf one var-6))
  (when var-8 (setf ldt var-8))
  (when var-10 (setf ldwork var-10)))
(cond
  ((> lastv k)
   (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10
      var-11 var-12)
     (zgemm "No transpose" "No transpose" lastc
      (f2cl-lib:int-sub lastv k)
      k (- one) work ldwork v ldv one c ldc)
     (declare (ignore var-0 var-1 var-3 var-5 var-6 var-8 var-11))
     (when var-2 (setf lastc var-2)) (when var-4 (setf k var-4))
     (when var-7 (setf ldwork var-7)) (when var-9 (setf ldv var-9))
     (when var-10 (setf one var-10))
     (when var-12 (setf ldc var-12))))))
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
     var-8 var-9 var-10)
    (ztrmm "Right" "Lower" "No transpose" "Unit" lastc k one
      (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add (f2cl-lib:int-sub lastv k) 1))
        ((1 ldv) (1 *)))
      v-%offset%)
    ldv work ldwork)
    (declare (ignore var-0 var-1 var-2 var-3 var-7 var-9))
    (when var-4 (setf lastc var-4)) (when var-5 (setf k var-5))
    (when var-6 (setf one var-6)) (when var-8 (setf ldv var-8))
    (when var-10 (setf ldwork var-10)))
  (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
    ((> j k) nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i lastc) nil)
        (tagbody
          (setf
            (f2cl-lib:fref c-%data%
              (i (f2cl-lib:int-add (f2cl-lib:int-sub
                lastv k) j)) ((1 ldc) (1 *)))
            c-%offset%)
          (-
            (f2cl-lib:fref c-%data%

```

```

        (i (f2cl-lib:int-add
            (f2cl-lib:int-sub lastv k) j))
        ((1 ldc) (1 *))
        c-%offset%)
        (f2cl-lib:fref work-%data% (i j)
          ((1 ldwork) (1 *)) work-%offset%)))
        label230))
        label240))))))
    (go end_label) end_label
    (return
     (values side trans direct storev nil nil k nil ldv nil ldt nil ldc nil
              ldwork))))))

```

zlarf LAPACK

— zlarf.input —

```

)set break resume
)sys rm -f zlarf.output
)spool zlarf.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlarf.help —

```

=====
zlarf examples
=====

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
SUBROUTINE ZLARF( SIDE, M, N, V, INCV, TAU, C, LDC, WORK )
```

```
.. Scalar Arguments ..
CHARACTER          SIDE
INTEGER            INCV, LDC, M, N
COMPLEX*16         TAU
..
.. Array Arguments ..
COMPLEX*16         C( LDC, * ), V( * ), WORK( * )
..
```

Purpose:

=====

ZLARF applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right. H is represented in the form

$$H = I - \tau * v * v^* H$$

where tau is a complex scalar and v is a complex vector.

If tau = 0, then H is taken to be the unit matrix.

To apply H**H, supply conjg(tau) instead tau.

Arguments:

=====

[in] SIDE

SIDE is CHARACTER*1
 = 'L': form $H * C$
 = 'R': form $C * H$

[in] M

M is INTEGER
 The number of rows of the matrix C.

[in] N

N is INTEGER
 The number of columns of the matrix C.

[in] V

V is COMPLEX*16 array, dimension
 (1 + (M-1)*abs(INCV)) if SIDE = 'L'
 or (1 + (N-1)*abs(INCV)) if SIDE = 'R'
 The vector v in the representation of H. V is not used if
 TAU = 0.

[in] INCV

INCV is INTEGER
 The increment between elements of v. INCV <> 0.

[in] TAU

TAU is COMPLEX*16
 The value tau in the representation of H.

[in,out] C

C is COMPLEX*16 array, dimension (LDC,N)
 On entry, the M-by-N matrix C.
 On exit, C is overwritten by the matrix $H * C$ if SIDE = 'L',
 or $C * H$ if SIDE = 'R'.

[in] LDC

LDC is INTEGER
 The leading dimension of the array C. LDC >= max(1,M).

[out] WORK

WORK is COMPLEX*16 array, dimension
 (N) if SIDE = 'L'
 or (M) if SIDE = 'R'

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlarf.f —

```

* =====
*      SUBROUTINE ZLARF( SIDE, M, N, V, INCV, TAU, C, LDC, WORK )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
* .. Scalar Arguments ..
*      CHARACTER          SIDE
*      INTEGER            INCV, LDC, M, N
*      COMPLEX*16         TAU
*
* ..
* .. Array Arguments ..
*      COMPLEX*16         C( LDC, * ), V( * ), WORK( * )
* ..
* =====
*
* .. Parameters ..
*      COMPLEX*16         ONE, ZERO
*      PARAMETER          ( ONE = ( 1.0D+0, 0.0D+0 ),
* $                        ZERO = ( 0.0D+0, 0.0D+0 ) )
* ..
* .. Local Scalars ..
*      LOGICAL            APPLYLEFT
*      INTEGER            I, LASTV, LASTC
* ..
* .. External Subroutines ..
*      EXTERNAL           ZGEMV, ZGERC
* ..
* .. External Functions ..
*      LOGICAL            LSAME
*      INTEGER            ILAZLR, ILAZLC
*      EXTERNAL           LSAME, ILAZLR, ILAZLC
* ..
* .. Executable Statements ..
*
*      APPLYLEFT = LSAME( SIDE, 'L' )
*      LASTV = 0
*      LASTC = 0
*      IF( TAU.NE.ZERO ) THEN
*
* Set up variables for scanning V.  LASTV begins pointing to the end
* of V.
*      IF( APPLYLEFT ) THEN
*          LASTV = M
*      ELSE
*          LASTV = N
*      END IF
*      IF( INCV.GT.0 ) THEN

```



```

        I = 1 + (LASTV-1) * INCV
    ELSE
        I = 1
    END IF
*   Look for the last non-zero row in V.
    DO WHILE( LASTV.GT.0 .AND. V( I ).EQ.ZERO )
        LASTV = LASTV - 1
        I = I - INCV
    END DO
    IF( APPLYLEFT ) THEN
*   Scan for the last non-zero column in C(1:lastv,:).
        LASTC = ILAZLC(LASTV, N, C, LDC)
    ELSE
*   Scan for the last non-zero row in C(:,1:lastv).
        LASTC = ILAZLR(M, LASTV, C, LDC)
    END IF
    END IF
*   Note that lastc.eq.0 renders the BLAS operations null; no special
*   case is needed at this level.
    IF( APPLYLEFT ) THEN
*
*       Form  H * C
*
        IF( LASTV.GT.0 ) THEN
*
*           w(1:lastc,1) := C(1:lastv,1:lastc)**H * v(1:lastv,1)
*
*           CALL ZGEMV( 'Conjugate transpose', LASTV, LASTC, ONE,
$               C, LDC, V, INCV, ZERO, WORK, 1 )
*
*           C(1:lastv,1:lastc) := C(...) - v(1:lastv,1) * w(1:lastc,1)**H
*
*           CALL ZGERC( LASTV, LASTC, -TAU, V, INCV, WORK, 1, C, LDC )
        END IF
    ELSE
*
*       Form  C * H
*
        IF( LASTV.GT.0 ) THEN
*
*           w(1:lastc,1) := C(1:lastc,1:lastv) * v(1:lastv,1)
*
*           CALL ZGEMV( 'No transpose', LASTC, LASTV, ONE, C, LDC,
$               V, INCV, ZERO, WORK, 1 )
*
*           C(1:lastc,1:lastv) := C(...) - w(1:lastc,1) * v(1:lastv,1)**H
*
*           CALL ZGERC( LASTC, LASTV, -TAU, WORK, 1, V, INCV, C, LDC )
        END IF
    END IF

```

```

      RETURN
*
*      End of ZLARF
*
      END

```

— LAPACK zlarf —

```

(let*
  ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16))
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (type (f2cl-lib:complex16) zero)
            (ignorable one zero))
  (defun zlarf (side m n v incv tau c ldc work)
    (declare (type (simple-array character (*)) side)
              (type (f2cl-lib:integer4) ldc incv n m)
              (type (array f2cl-lib:complex16 (*)) work c v)
              (type (f2cl-lib:complex16) tau))
    (f2cl-lib:with-multi-array-data
      ((v f2cl-lib:complex16 v-%data% v-%offset%)
       (c f2cl-lib:complex16 c-%data% c-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%)
       (side character side-%data% side-%offset%))
      (prog
        ((i 0) (lastv 0) (lastc 0) (applyleft nil))
        (declare (type (f2cl-lib:integer4) lastc lastv i)
                  (type f2cl-lib:logical applyleft))
        (setf applyleft
              (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
                (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
        (setf lastv 0) (setf lastc 0)
        (cond
          ((/= tau zero)
           (tagbody (cond (applyleft (setf lastv m)) (t (setf lastv n)))
                     (cond
                      ((> incv 0)
                       (setf i
                            (f2cl-lib:int-add 1
          (f2cl-lib:int-mul (f2cl-lib:int-sub lastv 1) incv))))
                      (t (setf i 1))))
                     label100000
                     (if
                      (not
                       (and (> lastv 0)
                            (= (f2cl-lib:fref v-%data% (i) ((1 *)) v-%offset%) zero)))
                      (go label100001)))

```

```

(setf lastv (f2cl-lib:int-sub lastv 1))
(setf i (f2cl-lib:int-sub i incv))
(go label100000) label100001
(cond (applyleft (setf lastc (ilazlc lastv n c ldc)))
      (t (setf lastc (ilazlr m lastv c ldc))))))
(cond
 (applyleft
  (cond
   ((> lastv 0)
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
      var-8 var-9 var-10)
     (zgemv "Conjugate transpose" lastv lastc one c
      ldc v incv zero work 1)
    (declare (ignore var-0 var-4 var-6 var-9 var-10))
    (when var-1 (setf lastv var-1)) (when var-2 (setf lastc var-2))
    (when var-3 (setf one var-3)) (when var-5 (setf ldc var-5))
    (when var-7 (setf incv var-7)) (when var-8 (setf zero var-8)))
    (multiple-value-bind
     (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
     (zgerc lastv lastc (- tau) v incv work 1 c ldc)
    (declare (ignore var-2 var-3 var-5 var-6 var-7))
    (when var-0 (setf lastv var-0)) (when var-1 (setf lastc var-1))
    (when var-4 (setf incv var-4)) (when var-8 (setf ldc var-8))))))
  (t
   (cond
    ((> lastv 0)
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
       var-8 var-9 var-10)
      (zgemv "No transpose" lastc lastv one c ldc v incv zero work 1)
     (declare (ignore var-0 var-4 var-6 var-9 var-10))
     (when var-1 (setf lastc var-1)) (when var-2 (setf lastv var-2))
     (when var-3 (setf one var-3)) (when var-5 (setf ldc var-5))
     (when var-7 (setf incv var-7)) (when var-8 (setf zero var-8)))
     (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
      (zgerc lastc lastv (- tau) work 1 v incv c ldc)
     (declare (ignore var-2 var-3 var-4 var-5 var-7))
     (when var-0 (setf lastc var-0)) (when var-1 (setf lastv var-1))
     (when var-6 (setf incv var-6)) (when var-8 (setf ldc var-8))))))
    (go end_label) end_label
    (return (values side nil nil nil incv nil nil ldc nil))))))

```

zlarfg LAPACK

— zlarfg.input —

```
)set break resume
)sys rm -f zlarfg.output
)spool zlarfg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zlarfg.help —

```
=====
zlarfg examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
SUBROUTINE ZLARFG( N, ALPHA, X, INCX, TAU )
```

```
.. Scalar Arguments ..
INTEGER              INCX, N
COMPLEX*16           ALPHA, TAU
..
.. Array Arguments ..
COMPLEX*16           X( * )
..
```

Purpose:
 =====

ZLARFG generates a complex elementary reflector H of order n, such

that

$$\begin{matrix} H^{**}H * (\text{alpha}) & = & (\text{beta}), & H^{**}H * H = I. \\ (\text{ x }) & & (\text{ 0 }) \end{matrix}$$

where alpha and beta are scalars, with beta real, and x is an (n-1)-element complex vector. H is represented in the form

$$H = I - \text{tau} * \begin{pmatrix} 1 \\ \text{v} \end{pmatrix} * \begin{pmatrix} 1 & \text{v}^{**}H \end{pmatrix},$$

where tau is a complex scalar and v is a complex (n-1)-element vector. Note that H is not hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Otherwise $1 \leq \text{real}(\text{tau}) \leq 2$ and $|\text{abs}(\text{tau}-1)| \leq 1$.

Arguments:

=====

[in] N

N is INTEGER
The order of the elementary reflector.

[in,out] ALPHA

ALPHA is COMPLEX*16
On entry, the value alpha.
On exit, it is overwritten with the value beta.

[in,out] X

X is COMPLEX*16 array, dimension
(1+(N-2)*abs(INCX))
On entry, the vector x.
On exit, it is overwritten with the vector v.

[in] INCX

INCX is INTEGER
The increment between elements of X. INCX > 0.

[out] TAU

TAU is COMPLEX*16
The value tau.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlzrfg.f —

```
* =====
*      SUBROUTINE ZLARFG( N, ALPHA, X, INCX, TAU )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          INCX, N
*      COMPLEX*16       ALPHA, TAU
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       X( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION ONE, ZERO
*      PARAMETER        ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
*      ..
*      .. Local Scalars ..
*      INTEGER          J, KNT
*      DOUBLE PRECISION ALPHI, ALPHR, BETA, RSAFMN, SAFMIN, XNORM
*
*      ..
*      .. External Functions ..
*      DOUBLE PRECISION DLAMCH, DLAPY3, DZNRM2
*      COMPLEX*16       ZLADIV
*      EXTERNAL         DLAMCH, DLAPY3, DZNRM2, ZLADIV
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC        ABS, DBLE, DCMPLX, DIMAG, SIGN
*
*      ..
```

```

*      .. External Subroutines ..
EXTERNAL          ZDSCAL, ZSCAL
*
*      ..
*      .. Executable Statements ..
*
      IF( N.LE.0 ) THEN
        TAU = ZERO
        RETURN
      END IF
*
      XNORM = DZNRM2( N-1, X, INCX )
      ALPHR = DBLE( ALPHA )
      ALPHI = DIMAG( ALPHA )
*
      IF( XNORM.EQ.ZERO .AND. ALPHI.EQ.ZERO ) THEN
*
*       H = I
*
*       TAU = ZERO
      ELSE
*
*       general case
*
      BETA = -SIGN( DLAPY3( ALPHR, ALPHI, XNORM ), ALPHR )
      SAFMIN = DLAMCH( 'S' ) / DLAMCH( 'E' )
      RSAFMN = ONE / SAFMIN
*
      KNT = 0
      IF( ABS( BETA ).LT.SAFMIN ) THEN
*
*       XNORM, BETA may be inaccurate; scale X and recompute them
*
10      CONTINUE
        KNT = KNT + 1
        CALL ZDSCAL( N-1, RSAFMN, X, INCX )
        BETA = BETA*RSAFMN
        ALPHI = ALPHI*RSAFMN
        ALPHR = ALPHR*RSAFMN
        IF( ABS( BETA ).LT.SAFMIN )
          $      GO TO 10
*
*       New BETA is at most 1, at least SAFMIN
*
      XNORM = DZNRM2( N-1, X, INCX )
      ALPHA = DCMLX( ALPHR, ALPHI )
      BETA = -SIGN( DLAPY3( ALPHR, ALPHI, XNORM ), ALPHR )
      END IF
      TAU = DCMLX( ( BETA-ALPHR ) / BETA, -ALPHI / BETA )
      ALPHA = ZLADIV( DCMLX( ONE ), ALPHA-BETA )
      CALL ZSCAL( N-1, ALPHA, X, INCX )

```

```

*
*      If ALPHA is subnormal, it may lose relative accuracy
*
      DO 20 J = 1, KNT
          BETA = BETA*SAFMIN
20      CONTINUE
          ALPHA = BETA
      END IF
*
      RETURN
*
*      End of ZLARFG
*
      END

```

— LAPACK zlarfg —

```

(let* ((one 1.0d0) (zero 0.0d0))
  (declare (type (double-float 1.0d0 1.0d0) one)
    (type (double-float 0.0d0 0.0d0) zero) (ignorable one zero))
  (defun zlarfg (n alpha x incx tau)
    (declare (type (f2cl-lib:integer4) incx n)
      (type (f2cl-lib:complex16) tau alpha)
      (type (array f2cl-lib:complex16 (*)) x))
    (f2cl-lib:with-multi-array-data
      ((x f2cl-lib:complex16 x-%data% x-%offset%))
      (prog
        ((alphi 0.0d0) (alphr 0.0d0) (beta 0.0d0) (rsafmn 0.0d0) (safmin 0.0d0)
          (xnrm 0.0d0) (j 0) (knt 0))
        (declare (type (double-float) xnrm safmin rsafmn beta alphr alphi)
          (type (f2cl-lib:integer4) knt j))
        (cond ((<= n 0) (setf tau (coerce zero 'f2cl-lib:complex16))
          (go end_label)))
        (setf xnrm
          (multiple-value-bind (ret-val var-0 var-1 var-2)
            (dznrm2 (f2cl-lib:int-sub n 1) x incx) (declare (ignore var-0 var-1))
            (when var-2 (setf incx var-2)) ret-val))
        (setf alphr (f2cl-lib:dbler alpha)) (setf alphi (f2cl-lib:dimag alpha))
        (cond
          ((and (= xnrm zero) (= alphi zero))
            (setf tau (coerce zero 'f2cl-lib:complex16)))
          (t (setf beta (- (f2cl-lib:sign (dlapy3 alphr alphi xnrm) alphr)))
            (setf safmin (/ (dlamch "S") (dlamch "E")))
            (setf rsafmn (/ one safmin))
            (setf knt 0)
            (cond

```



```

((< (abs beta) safmin)
 (tagbody label10 (setf knt (f2cl-lib:int-add knt 1))
  (multiple-value-bind (var-0 var-1 var-2 var-3)
    (zdscal (f2cl-lib:int-sub n 1) rsafmn x incx)
    (declare (ignore var-0 var-2)) (when var-1 (setf rsafmn var-1))
    (when var-3 (setf incx var-3)))
  (setf beta (* beta rsafmn)) (setf alphi (* alphi rsafmn))
  (setf alphi (* alphi rsafmn))
  (if (< (abs beta) safmin) (go label10))
  (setf xnorm
    (multiple-value-bind (ret-val var-0 var-1 var-2)
      (dznrm2 (f2cl-lib:int-sub n 1) x incx)
      (declare (ignore var-0 var-1))
      (when var-2 (setf incx var-2)) ret-val))
    (setf alpha (f2cl-lib:dcmplx alphi alphi))
    (setf beta
      (- (f2cl-lib:sign (dlapy3 alphi alphi xnorm) alphi))))))
(setf tau
  (f2cl-lib:dcmplx (/ (- beta alphi) beta) (/ (- alphi) beta)))
(setf alpha (zladi (f2cl-lib:dcmplx one) (- alpha beta)))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zscal (f2cl-lib:int-sub n 1) alpha x incx)
  (declare (ignore var-0 var-2))
  (when var-1 (setf alpha var-1)) (when var-3 (setf incx var-3)))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j knt) nil)
  (tagbody
    (setf beta (* beta safmin)) label20))
(setf alpha (coerce beta 'f2cl-lib:complex16)))
(go end_label) end_label (return (values nil alpha nil incx tau))))

```

zlarft LAPACK

— zlarft.input —

```

)set break resume
)sys rm -f zlarft.output
)spool zlarft.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlarft.help —

```
=====
zlarft examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLARFT( DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          DIRECT, STOREV
```

```
INTEGER           K, LDT, LDV, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16        T( LDT, * ), TAU( * ), V( LDV, * )
```

```
..
```

Purpose:

```
=====
```

ZLARFT forms the triangular factor T of a complex block reflector H of order n, which is defined as a product of k elementary reflectors.

If DIRECT = 'F', H = H(1) H(2) . . . H(k) and T is upper triangular;

If DIRECT = 'B', H = H(k) . . . H(2) H(1) and T is lower triangular.

If STOREV = 'C', the vector which defines the elementary reflector H(i) is stored in the i-th column of the array V, and

$$H = I - V * T * V^{**H}$$

If STOREV = 'R', the vector which defines the elementary reflector H(i) is stored in the i-th row of the array V, and

$$H = I - V^{**H} * T * V$$

Arguments:

=====

[in] DIRECT

DIRECT is CHARACTER*1

Specifies the order in which the elementary reflectors are multiplied to form the block reflector:

= 'F': $H = H(1) H(2) \dots H(k)$ (Forward)= 'B': $H = H(k) \dots H(2) H(1)$ (Backward)

[in] STOREV

STOREV is CHARACTER*1

Specifies how the vectors which define the elementary reflectors are stored (see also Further Details):

= 'C': columnwise

= 'R': rowwise

[in] N

N is INTEGER

The order of the block reflector H. $N \geq 0$.

[in] K

K is INTEGER

The order of the triangular factor T (= the number of elementary reflectors). $K \geq 1$.

[in,out] V

V is COMPLEX*16 array, dimension

(LDV,K) if STOREV = 'C'

(LDV,N) if STOREV = 'R'

The matrix V. See further details.

[in] LDV

LDV is INTEGER

The leading dimension of the array V.

If STOREV = 'C', $LDV \geq \max(1,N)$; if STOREV = 'R', $LDV \geq K$.

[in] TAU

TAU is COMPLEX*16 array, dimension (K)

TAU(i) must contain the scalar factor of the elementary reflector H(i).

[out] T

T is COMPLEX*16 array, dimension (LDT,K)
 The k by k triangular factor T of the block reflector.
 If DIRECT = 'F', T is upper triangular; if DIRECT = 'B', T is
 lower triangular. The rest of the array is not used.

[in] LDT

LDT is INTEGER
 The leading dimension of the array T. LDT >= K.

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:

=====

The shape of the matrix V and the storage of the vectors which define
 the H(i) is best illustrated by the following example with n = 5 and
 k = 3. The elements equal to 1 are not stored; the corresponding
 array elements are modified but restored on exit. The rest of the
 array is not used.

DIRECT = 'F' and STOREV = 'C':

$$V = \begin{pmatrix} 1 & & \\ v1 & 1 & \\ v1 & v2 & 1 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \end{pmatrix}$$

DIRECT = 'F' and STOREV = 'R':

$$V = \begin{pmatrix} 1 & v1 & v1 & v1 & v1 \\ & 1 & v2 & v2 & v2 \\ & & 1 & v3 & v3 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'C':

$$V = \begin{pmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ 1 & v2 & v3 \\ & 1 & v3 \\ & & 1 \end{pmatrix}$$

DIRECT = 'B' and STOREV = 'R':

$$V = \begin{pmatrix} v1 & v1 & 1 & & \\ v2 & v2 & v2 & 1 & \\ v3 & v3 & v3 & v3 & 1 \end{pmatrix}$$

—————>

— zlarft.f —

```

* =====
*      SUBROUTINE ZLARFT( DIRECT, STOREV, N, K, V, LDV, TAU, T, LDT )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          DIRECT, STOREV
*      INTEGER            K, LDT, LDV, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16         T( LDT, * ), TAU( * ), V( LDV, * )
*      ..
*
* =====
*
*      .. Parameters ..
*      COMPLEX*16         ONE, ZERO
*      PARAMETER          ( ONE = ( 1.0D+0, 0.0D+0 ),
*      $                  ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Local Scalars ..
*      INTEGER            I, J, PREVLASTV, LASTV
*      COMPLEX*16         VII
*
*      ..
*      .. External Subroutines ..
*      EXTERNAL           ZGEMV, ZLACGV, ZTRMV
*
*      ..
*      .. External Functions ..
*      LOGICAL            LSAME
*      EXTERNAL           LSAME
*
*      ..
*      .. Executable Statements ..
*
*      Quick return if possible
*
*      IF( N.EQ.0 )
*      $      RETURN
*
*      IF( LSAME( DIRECT, 'F' ) ) THEN
*          PREVLASTV = N
*          DO 20 I = 1, K
*              PREVLASTV = MAX( PREVLASTV, I )
*              IF( TAU( I ).EQ.ZERO ) THEN
*
*                  H(i) = I

```

```

*
      DO 10 J = 1, I
        T( J, I ) = ZERO
10    CONTINUE
      ELSE
*
*      general case
*
      VII = V( I, I )
      V( I, I ) = ONE
      IF( LSAME( STOREV, 'C' ) ) THEN
!        Skip any trailing zeros.
        DO LASTV = N, I+1, -1
          IF( V( LASTV, I ).NE.ZERO ) EXIT
        END DO
        J = MIN( LASTV, PREVLASTV )
*
*      T(1:i-1,i) := - tau(i) * V(i:j,1:i-1)**H * V(i:j,i)
*
        CALL ZGEMV( 'Conjugate transpose', J-I+1, I-1,
$          -TAU( I ), V( I, 1 ), LDV, V( I, I ), 1,
$          ZERO, T( 1, I ), 1 )
      ELSE
!        Skip any trailing zeros.
        DO LASTV = N, I+1, -1
          IF( V( I, LASTV ).NE.ZERO ) EXIT
        END DO
        J = MIN( LASTV, PREVLASTV )
*
*      T(1:i-1,i) := - tau(i) * V(1:i-1,i:j) * V(i,i:j)**H
*
        IF( I.LT.J )
$          CALL ZLACGV( J-I, V( I, I+1 ), LDV )
        CALL ZGEMV( 'No transpose', I-1, J-I+1, -TAU( I ),
$          V( 1, I ), LDV, V( I, I ), LDV, ZERO,
$          T( 1, I ), 1 )
        IF( I.LT.J )
$          CALL ZLACGV( J-I, V( I, I+1 ), LDV )
      END IF
      V( I, I ) = VII
*
*      T(1:i-1,i) := T(1:i-1,1:i-1) * T(1:i-1,i)
*
      CALL ZTRMV( 'Upper', 'No transpose', 'Non-unit', I-1, T,
$          LDT, T( 1, I ), 1 )
      T( I, I ) = TAU( I )
      IF( I.GT.1 ) THEN
        PREVLASTV = MAX( PREVLASTV, LASTV )
      ELSE
        PREVLASTV = LASTV

```

```

                END IF
            END IF
20    CONTINUE
    ELSE
        PREVLASTV = 1
        DO 40 I = K, 1, -1
            IF( TAU( I ).EQ.ZERO ) THEN
*
*           H(i) = I
*
                DO 30 J = I, K
                    T( J, I ) = ZERO
30                CONTINUE
            ELSE
*
*           general case
*
                IF( I.LT.K ) THEN
                    IF( LSAME( STOREV, 'C' ) ) THEN
                        VII = V( N-K+I, I )
                        V( N-K+I, I ) = ONE
!                        Skip any leading zeros.
                        DO LASTV = 1, I-1
                            IF( V( LASTV, I ).NE.ZERO ) EXIT
                        END DO
                        J = MAX( LASTV, PREVLASTV )
*
*                        T(i+1:k,i) :=
*                        - tau(i) * V(j:n-k+i,i+1:k)**H * V(j:n-k+i,i)
*
                        CALL ZGEMV( 'Conjugate transpose', N-K+I-J+1, K-I,
$                            -TAU( I ), V( J, I+1 ), LDV, V( J, I ),
$                            1, ZERO, T( I+1, I ), 1 )
                        V( N-K+I, I ) = VII
                    ELSE
                        VII = V( I, N-K+I )
                        V( I, N-K+I ) = ONE
!                        Skip any leading zeros.
                        DO LASTV = 1, I-1
                            IF( V( I, LASTV ).NE.ZERO ) EXIT
                        END DO
                        J = MAX( LASTV, PREVLASTV )
*
*                        T(i+1:k,i) :=
*                        - tau(i) * V(i+1:k,j:n-k+i) * V(i,j:n-k+i)**H
*
                        CALL ZLACGV( N-K+I-1-J+1, V( I, J ), LDV )
                        CALL ZGEMV( 'No transpose', K-I, N-K+I-J+1,
$                            -TAU( I ), V( I+1, J ), LDV, V( I, J ), LDV,
$                            ZERO, T( I+1, I ), 1 )

```

```

        CALL ZLACGV( N-K+I-1-J+1, V( I, J ), LDV )
        V( I, N-K+I ) = VII
    END IF

*
*      T(i+1:k,i) := T(i+1:k,i+1:k) * T(i+1:k,i)
*
        CALL ZTRMV( 'Lower', 'No transpose', 'Non-unit', K-I,
$          T( I+1, I+1 ), LDT, T( I+1, I ), 1 )
        IF( I.GT.1 ) THEN
            PREVLASTV = MIN( PREVLASTV, LASTV )
        ELSE
            PREVLASTV = LASTV
        END IF
    END IF
    T( I, I ) = TAU( I )
END IF
40    CONTINUE
    END IF
    RETURN

*
*      End of ZLARFT
*
    END

```

— LAPACK zlarft —

```

(let*
  ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16))
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (type (f2cl-lib:complex16) zero)
   (ignorable one zero))
  (defun zlarft (direct storev n k v ldv tau t$ ldt)
    (declare (type (simple-array character (*)) storev direct)
     (type (f2cl-lib:integer4) ldt ldv k n)
     (type (array f2cl-lib:complex16 (*)) t$ tau v))
    (f2cl-lib:with-multi-array-data
      ((v f2cl-lib:complex16 v-%data% v-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
       (direct character direct-%data% direct-%offset%)
       (storev character storev-%data% storev-%offset%))
      (prog
        ((vii #C(0.0d0 0.0d0)) (i 0) (j 0) (prevlastv 0) (lastv 0))
        (declare (type (f2cl-lib:complex16) vii)
         (type (f2cl-lib:integer4) lastv prevlastv j i))
        (if (= n 0) (go end_label))

```



```

(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame direct "F")
    (declare (ignore var-1)) (when var-0 (setf direct var-0)) ret-val)
   (setf prevlastv n)
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
     (> i k) nil)
   (tagbody
     (setf prevlastv
       (max (the f2cl-lib:integer4 prevlastv)
            (the f2cl-lib:integer4 i)))
     (cond
       ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          (> j i) nil)
        (tagbody
          (setf (f2cl-lib:fref t$-%data% (j i)
            ((1 ldt) (1 *)) t$-%offset%) zero)
            label10)))
       (t (setf vii (f2cl-lib:fref v-%data% (i i)
            ((1 ldv) (1 *)) v-%offset%))
          (setf (f2cl-lib:fref v-%data% (i i)
            ((1 ldv) (1 *)) v-%offset%) one)
          (cond
            ((multiple-value-bind (ret-val var-0 var-1)
              (lsame storev "C")
              (declare (ignore var-1))
              (when var-0 (setf storev var-0)) ret-val)
             (f2cl-lib:fdo (lastv n (f2cl-lib:int-add lastv
              (f2cl-lib:int-sub 1)))
              (>
                lastv (f2cl-lib:int-add i 1))
                nil)
             (tagbody
               (if
                 (/= (f2cl-lib:fref v-%data% (lastv i)
                  ((1 ldv) (1 *)) v-%offset%)
                  zero)
                 (go f2cl-lib::exit))
                 label100000)))
            (setf j
              (min (the f2cl-lib:integer4 lastv)
                   (the f2cl-lib:integer4 prevlastv)))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5
               var-6 var-7 var-8 var-9 var-10)
              (zgemv "Conjugate transpose"
                (f2cl-lib:int-add (f2cl-lib:int-sub j i) 1)
                (f2cl-lib:int-sub i 1)
                (- (f2cl-lib:fref tau-%data% (i) ((1 *))
                  tau-%offset%))

```

```

(f2cl-lib:array-slice v-%data%
 f2cl-lib:complex16 (i 1)
 ((1 ldv) (1 *)) v-%offset%)
ldv
(f2cl-lib:array-slice v-%data%
 f2cl-lib:complex16 (i i)
 ((1 ldv) (1 *)) v-%offset%)
1 zero
(f2cl-lib:array-slice t$-%data%
 f2cl-lib:complex16 (1 i)
 ((1 ldt) (1 *)) t$-%offset%)
1)
(declare
 (ignore var-0 var-1 var-2 var-3 var-4
          var-6 var-7 var-9 var-10))
(when var-5 (setf ldv var-5))
(when var-8 (setf zero var-8)))
(t
 (f2cl-lib:fdo (lastv n (f2cl-lib:int-add lastv
                      (f2cl-lib:int-sub 1)))
  ((>
    lastv (f2cl-lib:int-add i 1))
   nil)
 (tagbody
  (if
   (/= (f2cl-lib:fref v-%data% (i lastv)
                     ((1 ldv) (1 *)) v-%offset%)
    zero)
   (go f2cl-lib::exit))
  label100001))
(setf j
 (min (the f2cl-lib:integer4 lastv)
      (the f2cl-lib:integer4 prevlastv)))
(if (< i j)
 (zlacgv (f2cl-lib:int-sub j i)
  (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
    (i (f2cl-lib:int-add i 1))
    ((1 ldv) (1 *)) v-%offset%)
  ldv))
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4
  var-5 var-6 var-7 var-8 var-9 var-10)
 (zgemv "No transpose" (f2cl-lib:int-sub i 1)
  (f2cl-lib:int-add (f2cl-lib:int-sub j i) 1)
  (- (f2cl-lib:fref tau-%data% (i) ((1 *))
    tau-%offset%))
  (f2cl-lib:array-slice v-%data%
    f2cl-lib:complex16 (1 i)
    ((1 ldv) (1 *)) v-%offset%)
  ldv

```

```

(f2cl-lib:array-slice v-%data%
  f2cl-lib:complex16 (i i)
  ((1 ldv) (1 *)) v-%offset%)
ldv zero
(f2cl-lib:array-slice t$-%data%
  f2cl-lib:complex16 (1 i)
  ((1 ldt) (1 *)) t$-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-3
              var-4 var-6 var-9 var-10))
(when var-5 (setf ldv var-5))
(when var-7 (setf ldv var-7))
(when var-8 (setf zero var-8))
(if (< i j)
  (zlacgv (f2cl-lib:int-sub j i)
    (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
      (i (f2cl-lib:int-add i 1))
      ((1 ldv) (1 *)) v-%offset%)
    ldv))))
(setf (f2cl-lib:fref v-%data% (i i)
  ((1 ldv) (1 *)) v-%offset%) vii)
(multiple-value-bind (var-0 var-1 var-2 var-3
  var-4 var-5 var-6 var-7)
  (ztrmv "Upper" "No transpose" "Non-unit"
    (f2cl-lib:int-sub i 1) t$ ldt
    (f2cl-lib:array-slice t$-%data%
      f2cl-lib:complex16 (1 i)
      ((1 ldt) (1 *)) t$-%offset%)
    1)
  (declare (ignore var-0 var-1 var-2 var-3
                    var-4 var-6 var-7))
  (when var-5 (setf ldt var-5)))
(setf (f2cl-lib:fref t$-%data% (i i)
  ((1 ldt) (1 *)) t$-%offset%)
  (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
(cond
  ((> i 1)
    (setf prevlastv
      (max (the f2cl-lib:integer4 prevlastv)
        (the f2cl-lib:integer4 lastv))))
    (t (setf prevlastv lastv))))
label20)))
(t (setf prevlastv 1)
  (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    ((> i 1) nil)
    (tagbody
      (cond
        ((= (f2cl-lib:fref tau (i) ((1 *))) zero)
          (f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
            ((> j k) nil)

```

```

(tagbody
  (setf (f2cl-lib:fref t$-%data% (j i)
    ((1 ldt) (1 *)) t$-%offset%) zero)
    label30)))
(t
  (cond
    ((< i k)
      (cond
        ((multiple-value-bind (ret-val var-0 var-1)
          (lsame storev "C")
          (declare (ignore var-1))
          (when var-0 (setf storev var-0)) ret-val)
          (setf vii
            (f2cl-lib:fref v-%data%
              ((f2cl-lib:int-add (f2cl-lib:int-sub n k) i) i)
              ((1 ldv) (1 *))
              v-%offset%))
            (setf
              (f2cl-lib:fref v-%data%
                ((f2cl-lib:int-add (f2cl-lib:int-sub n k) i) i)
                ((1 ldv) (1 *))
                v-%offset%)
              one)
            (f2cl-lib:fdo (lastv 1 (f2cl-lib:int-add lastv 1))
              (> lastv
                (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
                nil)
            (tagbody
              (if
                (/= (f2cl-lib:fref v-%data% (lastv i)
                  ((1 ldv) (1 *)) v-%offset%)
                  zero)
                (go f2cl-lib::exit))
                label1100002))
            (setf j
              (max (the f2cl-lib:integer4 lastv)
                (the f2cl-lib:integer4 prevlastv)))
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5
                var-6 var-7 var-8 var-9 var-10)
              (zgemv "Conjugate transpose"
                (f2cl-lib:int-add
                  (f2cl-lib:int-sub (f2cl-lib:int-add
                    (f2cl-lib:int-sub n k) i) j)
                    1)
                (f2cl-lib:int-sub k i)
                (- (f2cl-lib:fref tau-%data% (i)
                  ((1 *) tau-%offset%))
                  (f2cl-lib:array-slice v-%data% f2cl-lib:complex16
                    (j (f2cl-lib:int-add i 1))

```

```

                ((1 ldv) (1 *)) v-%offset%)
ldv
(f2cl-lib:array-slice v-%data%
 f2cl-lib:complex16 (j i)
 ((1 ldv) (1 *)) v-%offset%)
1 zero
(f2cl-lib:array-slice t$-%data%
 f2cl-lib:complex16 ((+ i 1) i)
 ((1 ldt) (1 *)) t$-%offset%)
1)
(declare
 (ignore var-0 var-1 var-2 var-3 var-4
        var-6 var-7 var-9 var-10))
(when var-5 (setf ldv var-5))
(when var-8 (setf zero var-8))
(setf
 (f2cl-lib:fref v-%data%
 ((f2cl-lib:int-add (f2cl-lib:int-sub n k) i) i)
 ((1 ldv) (1 *))
 v-%offset%)
 vii))
(t
 (setf vii
 (f2cl-lib:fref v-%data%
 (i (f2cl-lib:int-add (f2cl-lib:int-sub n k) i))
 ((1 ldv) (1 *))
 v-%offset%))
 (setf
 (f2cl-lib:fref v-%data%
 (i (f2cl-lib:int-add (f2cl-lib:int-sub n k) i))
 ((1 ldv) (1 *))
 v-%offset%)
 one)
 (f2cl-lib:fdo (lastv 1 (f2cl-lib:int-add lastv 1))
 (> lastv
  (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
  nil)
 (tagbody
  (if
   (/= (f2cl-lib:fref v-%data% (i lastv)
   ((1 ldv) (1 *)) v-%offset%)
   zero)
   (go f2cl-lib::exit))
  label100003))
(setf j
 (max (the f2cl-lib:integer4 lastv)
 (the f2cl-lib:integer4 prevlastv)))
(zlacgv
 (f2cl-lib:int-add
 (f2cl-lib:int-sub (f2cl-lib:int-add

```

```

(f2cl-lib:int-sub n k) i) 1 j)
1)
(f2cl-lib:array-slice v-%data%
 f2cl-lib:complex16 (i j)
 ((1 ldv) (1 *)) v-%offset%)
ldv)
(multiple-value-bind
 (var-0 var-1 var-2 var-3 var-4 var-5 var-6
 var-7 var-8 var-9 var-10)
 (zgemv "No transpose" (f2cl-lib:int-sub k i)
 (f2cl-lib:int-add
  (f2cl-lib:int-sub (f2cl-lib:int-add
   (f2cl-lib:int-sub n k) i) j)
  1)
 (- (f2cl-lib:fref tau-%data% (i)
   ((1 *)) tau-%offset%))
 (f2cl-lib:array-slice v-%data%
  f2cl-lib:complex16 ((+ i 1) j)
  ((1 ldv) (1 *)) v-%offset%)
ldv
 (f2cl-lib:array-slice v-%data%
  f2cl-lib:complex16 (i j)
  ((1 ldv) (1 *)) v-%offset%)
ldv zero
 (f2cl-lib:array-slice t$-%data%
  f2cl-lib:complex16 ((+ i 1) i)
  ((1 ldt) (1 *)) t$-%offset%)
1)
(declare (ignore var-0 var-1 var-2 var-3
                var-4 var-6 var-9 var-10))
(when var-5 (setf ldv var-5))
(when var-7 (setf ldv var-7))
(when var-8 (setf zero var-8))
(zlacgv
 (f2cl-lib:int-add
  (f2cl-lib:int-sub
   (f2cl-lib:int-add (f2cl-lib:int-sub n k) i) 1 j)
  1)
 (f2cl-lib:array-slice v-%data%
  f2cl-lib:complex16 (i j)
  ((1 ldv) (1 *)) v-%offset%)
ldv)
(setf
 (f2cl-lib:fref v-%data%
  (i (f2cl-lib:int-add
    (f2cl-lib:int-sub n k) i)) ((1 ldv) (1 *))
  v-%offset%)
vii)))
(multiple-value-bind (var-0 var-1 var-2 var-3
                    var-4 var-5 var-6 var-7)

```

```

(ztrmv "Lower" "No transpose" "Non-unit"
  (f2cl-lib:int-sub k i)
  (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
    ((+ i 1) (f2cl-lib:int-add i 1))
    ((1 ldt) (1 *)) t$-%offset%)
  ldt
  (f2cl-lib:array-slice t$-%data%
    f2cl-lib:complex16 ((+ i 1) i)
    ((1 ldt) (1 *)) t$-%offset%)
  1)
(declare (ignore var-0 var-1 var-2 var-3
  var-4 var-6 var-7))
(when var-5 (setf ldt var-5)))
(cond
  (> i 1)
  (setf prevlastv
    (min (the f2cl-lib:integer4 prevlastv)
      (the f2cl-lib:integer4 lastv))))
  (t (setf prevlastv lastv))))
(setf (f2cl-lib:fref t$-%data% (i i)
  ((1 ldt) (1 *)) t$-%offset%)
  (f2cl-lib:fref tau-%data% (i) ((1 *) tau-%offset%)))
label40)))
(go end_label) end_label
(return (values direct storev nil nil nil ldv nil nil ldt))))))

```

zlartg LAPACK

— zlartg.input —

```

)set break resume
)sys rm -f zlartg.output
)spool zlartg.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlartg.help —

=====

zlartg examples

Man Page Details

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
SUBROUTINE ZLARTG( F, G, CS, SN, R )
```

```
.. Scalar Arguments ..
DOUBLE PRECISION    CS
COMPLEX*16          F, G, R, SN
..
```

Purpose:

ZLARTG generates a plane rotation so that

$$\begin{bmatrix} CS & SN \\ -SN & CS \end{bmatrix} \cdot \begin{bmatrix} F \\ G \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{where } CS^2 + |SN|^2 = 1.$$

This is a faster version of the BLAS1 routine ZROTG, except for the following differences:

F and G are unchanged on return.

If G=0, then CS=1 and SN=0.

If F=0, then CS=0 and SN is chosen so that R is real.

Arguments:

[in] F

F is COMPLEX*16

The first component of vector to be rotated.

[in] G

G is COMPLEX*16

The second component of vector to be rotated.

[out] CS

CS is DOUBLE PRECISION
The cosine of the rotation.

[out] SN

SN is COMPLEX*16
The sine of the rotation.

[out] R

R is COMPLEX*16
The nonzero component of the rotated vector.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

Further Details:
=====

3-5-96 - Modified with a new algorithm by W. Kahan and J. Demmel

This version has a few statements commented out for thread safety
(machine parameters are computed on each entry). 10 feb 03, SJH.

— zlartg.f —

```
* =====
* SUBROUTINE ZLARTG( F, G, CS, SN, R )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
* November 2011
*
* .. Scalar Arguments ..
* DOUBLE PRECISION    CS
* COMPLEX*16          F, G, R, SN
```

```

*      ..
*
*      =====
*
*      .. Parameters ..
      DOUBLE PRECISION    TWO, ONE, ZERO
      PARAMETER            ( TWO = 2.0D+0, ONE = 1.0D+0, ZERO = 0.0D+0 )
      COMPLEX*16           CZERO
      PARAMETER            ( CZERO = ( 0.0D+0, 0.0D+0 ) )
*
*      ..
*      .. Local Scalars ..
*      LOGICAL             FIRST
      INTEGER              COUNT, I
      DOUBLE PRECISION     D, DI, DR, EPS, F2, F2S, G2, G2S, SAFMIN,
$      SAFMN2, SAFMX2, SCALE
      COMPLEX*16           FF, FS, GS
*
*      ..
*      .. External Functions ..
      DOUBLE PRECISION     DLAMCH, DLAPY2
      EXTERNAL              DLAMCH, DLAPY2
*
*      ..
*      .. Intrinsic Functions ..
      INTRINSIC             ABS, DBLE, DCMPLX, DCONJG, DIMAG, INT, LOG,
$      MAX, SQRT
*
*      ..
*      .. Statement Functions ..
      DOUBLE PRECISION     ABS1, ABSSQ
*
*      ..
*      .. Save statement ..
*      SAVE                 FIRST, SAFMX2, SAFMIN, SAFMN2
*
*      ..
*      .. Data statements ..
*      DATA                FIRST / .TRUE. /
*
*      ..
*      .. Statement Function definitions ..
      ABS1( FF ) = MAX( ABS( DBLE( FF ) ), ABS( DIMAG( FF ) ) )
      ABSSQ( FF ) = DBLE( FF )**2 + DIMAG( FF )**2
*
*      ..
*      .. Executable Statements ..
*
*      IF( FIRST ) THEN
          SAFMIN = DLAMCH( 'S' )
          EPS = DLAMCH( 'E' )
          SAFMN2 = DLAMCH( 'B' )**INT( LOG( SAFMIN / EPS ) /
$          LOG( DLAMCH( 'B' ) ) ) / TWO )
          SAFMX2 = ONE / SAFMN2
          FIRST = .FALSE.
*
*      END IF
      SCALE = MAX( ABS1( F ), ABS1( G ) )
      FS = F

```

```

      GS = G
      COUNT = 0
      IF( SCALE.GE.SAFMX2 ) THEN
10    CONTINUE
        COUNT = COUNT + 1
        FS = FS*SAFMN2
        GS = GS*SAFMN2
        SCALE = SCALE*SAFMN2
        IF( SCALE.GE.SAFMX2 )
$      GO TO 10
      ELSE IF( SCALE.LE.SAFMN2 ) THEN
        IF( G.EQ.CZERO ) THEN
          CS = ONE
          SN = CZERO
          R = F
          RETURN
        END IF
20    CONTINUE
        COUNT = COUNT - 1
        FS = FS*SAFMX2
        GS = GS*SAFMX2
        SCALE = SCALE*SAFMX2
        IF( SCALE.LE.SAFMN2 )
$      GO TO 20
      END IF
      F2 = ABSSQ( FS )
      G2 = ABSSQ( GS )
      IF( F2.LE.MAX( G2, ONE )*SAFMIN ) THEN
*
*      This is a rare case: F is very small.
*
      IF( F.EQ.CZERO ) THEN
        CS = ZERO
        R = DLAPY2( DBLE( G ), DIMAG( G ) )
*      Do complex/real division explicitly with two real divisions
        D = DLAPY2( DBLE( GS ), DIMAG( GS ) )
        SN = DCMPLX( DBLE( GS ) / D, -DIMAG( GS ) / D )
        RETURN
      END IF
      F2S = DLAPY2( DBLE( FS ), DIMAG( FS ) )
*      G2 and G2S are accurate
*      G2 is at least SAFMIN, and G2S is at least SAFMN2
      G2S = SQRT( G2 )
*      Error in CS from underflow in F2S is at most
*      UNFL / SAFMN2 .lt. sqrt(UNFL*EPS) .lt. EPS
*      If MAX(G2,ONE)=G2, then F2 .lt. G2*SAFMIN,
*      and so CS .lt. sqrt(SAFMIN)
*      If MAX(G2,ONE)=ONE, then F2 .lt. SAFMIN
*      and so CS .lt. sqrt(SAFMIN)/SAFMN2 = sqrt(EPS)
*      Therefore, CS = F2S/G2S / sqrt( 1 + (F2S/G2S)**2 ) = F2S/G2S

```

```

      CS = F2S / G2S
*      Make sure abs(FF) = 1
*      Do complex/real division explicitly with 2 real divisions
      IF( ABS1( F ).GT.ONE ) THEN
        D = DLAPY2( DBLE( F ), DIMAG( F ) )
        FF = DCMPLX( DBLE( F ) / D, DIMAG( F ) / D )
      ELSE
        DR = SAFMX2*DBLE( F )
        DI = SAFMX2*DIMAG( F )
        D = DLAPY2( DR, DI )
        FF = DCMPLX( DR / D, DI / D )
      END IF
      SN = FF*DCMPLX( DBLE( GS ) / G2S, -DIMAG( GS ) / G2S )
      R = CS*F + SN*G
    ELSE
*
*      This is the most common case.
*      Neither F2 nor F2/G2 are less than SAFMIN
*      F2S cannot overflow, and it is accurate
*
      F2S = SQRT( ONE+G2 / F2 )
*      Do the F2S(real)*FS(complex) multiply with two real multiplies
      R = DCMPLX( F2S*DBLE( FS ), F2S*DIMAG( FS ) )
      CS = ONE / F2S
      D = F2 + G2
*      Do complex/real division explicitly with two real divisions
      SN = DCMPLX( DBLE( R ) / D, DIMAG( R ) / D )
      SN = SN*DCONJG( GS )
      IF( COUNT.NE.0 ) THEN
        IF( COUNT.GT.0 ) THEN
          DO 30 I = 1, COUNT
            R = R*SAFMX2
30          CONTINUE
        ELSE
          DO 40 I = 1, -COUNT
            R = R*SAFMN2
40          CONTINUE
        END IF
      END IF
      END IF
      RETURN
*
*      End of ZLARTG
*
      END

```

— LAPACK zlarg —

```

(let*
  ((two 2.0d0) (one 1.0d0) (zero 0.0d0)
   (czero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (double-float 2.0d0 2.0d0) two)
            (type (double-float 1.0d0 1.0d0) one) (type (double-float 0.0d0 0.0d0) zero)
            (type (f2cl-lib:complex16) czero) (ignorable two one zero czero))
  (defun zlarg (f g cs sn r)
    (declare (type (f2cl-lib:complex16) r sn g f) (type (double-float) cs))
    (labels
      ((abs1 (ff) (max (abs (f2cl-lib:dblr ff)) (abs (f2cl-lib:dimag ff)))))
      (abssq (ff) (+ (expt (f2cl-lib:dblr ff) 2) (expt (f2cl-lib:dimag ff) 2))))
    (declare
      (ftype (function (f2cl-lib:complex16) (values double-float &rest t)) abs1))
    (declare
      (ftype (function (f2cl-lib:complex16) (values double-float &rest t))
              abssq))
    (prog
      ((ff #C(0.0d0 0.0d0)) (fs #C(0.0d0 0.0d0)) (gs #C(0.0d0 0.0d0)) (d 0.0d0)
       (di 0.0d0) (dr 0.0d0) (eps 0.0d0) (f2 0.0d0) (f2s 0.0d0) (g2 0.0d0)
       (g2s 0.0d0) (safmin 0.0d0) (safmn2 0.0d0) (safmx2 0.0d0) (scale 0.0d0)
       (i 0) (count$ 0))
      (declare (type (f2cl-lib:complex16) gs fs ff)
                (type (double-float) scale safmx2 safmn2 safmin g2s g2 f2s f2 eps dr di d)
                (type (f2cl-lib:integer4) count$ i))
      (setf safmin (dlamch "S")) (setf eps (dlamch "E"))
      (setf safmn2
        (expt (dlamch "B")
          (f2cl-lib:int
            (/ (/ (f2cl-lib:flog (/ safmin eps)) (f2cl-lib:flog (dlamch "B")))
              two))))
      (setf safmx2 (/ one safmn2)) (setf scale (max (abs1 f) (abs1 g)))
      (setf fs f) (setf gs g) (setf count$ 0)
      (cond
        ((>= scale safmx2)
         (tagbody label10 (setf count$ (f2cl-lib:int-add count$ 1))
                   (setf fs (* fs safmn2)) (setf gs (* gs safmn2))
                   (setf scale (* scale safmn2)) (if (>= scale safmx2) (go label10))))
        ((<= scale safmn2)
         (tagbody
           (cond
             ((= g czero) (setf cs one) (setf sn czero) (setf r f) (go end_label)))
           label20 (setf count$ (f2cl-lib:int-sub count$ 1))
                   (setf fs (* fs safmx2)) (setf gs (* gs safmx2))
                   (setf scale (* scale safmx2)) (if (<= scale safmn2) (go label20))))))
      (setf f2 (abssq fs)) (setf g2 (abssq gs))
      (cond
        ((<= f2 (* (max g2 one) safmin))
         (cond

```

```

(= f czero) (setf cs zero)
(setf r
  (coerce (dlapy2 (f2cl-lib:double g) (f2cl-lib:dimag g))
    'f2cl-lib:complex16))
(setf d (dlapy2 (f2cl-lib:double gs) (f2cl-lib:dimag gs)))
(setf sn
  (f2cl-lib:dcmplx (/ (f2cl-lib:double gs) d)
    (/ (- (f2cl-lib:dimag gs)) d)))
(go end_label)))
(setf f2s (dlapy2 (f2cl-lib:double fs) (f2cl-lib:dimag fs)))
(setf g2s (f2cl-lib:fsqrt g2)) (setf cs (/ f2s g2s))
(cond
  (> (abs1 f) one) (setf d (dlapy2 (f2cl-lib:double f) (f2cl-lib:dimag f)))
  (setf ff
    (f2cl-lib:dcmplx (/ (f2cl-lib:double f) d) (/ (f2cl-lib:dimag f) d))))
(t (setf dr (* safmx2 (f2cl-lib:double f)))
  (setf di (* safmx2 (f2cl-lib:dimag f)))
  (setf d
    (multiple-value-bind (ret-val var-0 var-1) (dlapy2 dr di)
      (declare (ignore)) (when var-0 (setf dr var-0))
      (when var-1 (setf di var-1)) ret-val))
  (setf ff (f2cl-lib:dcmplx (/ dr d) (/ di d))))))
(setf sn
  (* ff
    (f2cl-lib:dcmplx (/ (f2cl-lib:double gs) g2s)
      (/ (- (f2cl-lib:dimag gs)) g2s))))
(setf r (+ (* cs f) (* sn g)))
(t (setf f2s (f2cl-lib:fsqrt (+ one (/ g2 f2))))
  (setf r
    (f2cl-lib:dcmplx (* f2s (f2cl-lib:double fs))
      (* f2s (f2cl-lib:dimag fs))))
  (setf cs (/ one f2s)) (setf d (+ f2 g2))
  (setf sn
    (f2cl-lib:dcmplx (/ (f2cl-lib:double r) d) (/ (f2cl-lib:dimag r) d)))
  (setf sn (coerce (* sn (f2cl-lib:dconjg gs)) 'f2cl-lib:complex16))
  (cond
    ((/= count$ 0)
      (cond
        (> count$ 0)
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i count$) nil)
          (tagbody
            (setf r (* r safmx2)) label30)))
      (t
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i
            (f2cl-lib:int-sub count$))
            nil)
          (tagbody (setf r (* r safmn2)) label40))))))
  (go end_label) end_label (return (values nil nil cs sn r))))))

```

zlascl LAPACK

— zlascl.input —

```
)set break resume
)sys rm -f zlascl.output
)spool zlascl.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zlascl.help —

```
=====
zlascl examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLASCL( TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          TYPE
INTEGER            INFO, KL, KU, LDA, M, N
DOUBLE PRECISION   CFROM, CTO
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16         A( LDA, * )
```

```
..
```

Purpose:

=====

ZLASCL multiplies the M by N complex matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result $CTO * A(I,J) / CFROM$ does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Arguments:

=====

[in] TYPE

TYPE is CHARACTER*1

TYPE indices the storage type of the input matrix.

= 'G': A is a full matrix.

= 'L': A is a lower triangular matrix.

= 'U': A is an upper triangular matrix.

= 'H': A is an upper Hessenberg matrix.

= 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored.

= 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored.

= 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See ZGBTRF for storage details.

[in] KL

KL is INTEGER

The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

[in] KU

KU is INTEGER

The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

[in] CFROM

CFROM is DOUBLE PRECISION

[in] CTO

CTO is DOUBLE PRECISION

The matrix A is multiplied by CTO/CFROM. A(I,J) is computed

without over/underflow if the final result $CTO * A(I,J) / CFROM$ can be represented without over/underflow. $CFROM$ must be nonzero.

[in] M

M is INTEGER
The number of rows of the matrix A. $M \geq 0$.

[in] N

N is INTEGER
The number of columns of the matrix A. $N \geq 0$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)
The matrix to be multiplied by $CTO/CFROM$. See TYPE for the storage type.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

[out] INFO

INFO is INTEGER
0 - successful exit
<0 - if $INFO = -i$, the i-th argument had an illegal value.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zlascl.f —

```
* =====
* SUBROUTINE ZLASCL( TYPE, KL, KU, CFROM, CTO, M, N, A, LDA, INFO )
*
```

```

* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*   November 2011
*
*   .. Scalar Arguments ..
*   CHARACTER          TYPE
*   INTEGER            INFO, KL, KU, LDA, M, N
*   DOUBLE PRECISION   CFROM, CTO
*
*   ..
*   .. Array Arguments ..
*   COMPLEX*16         A( LDA, * )
*   ..
*
*   =====
*
*   .. Parameters ..
*   DOUBLE PRECISION   ZERO, ONE
*   PARAMETER          ( ZERO = 0.0D0, ONE = 1.0D0 )
*
*   ..
*   .. Local Scalars ..
*   LOGICAL            DONE
*   INTEGER            I, ITYPE, J, K1, K2, K3, K4
*   DOUBLE PRECISION   BIGNUM, CFROM1, CFROMC, CTO1, CTOC, MUL, SMLNUM
*
*   ..
*   .. External Functions ..
*   LOGICAL            LSAME, DISNAN
*   DOUBLE PRECISION   DLAMCH
*   EXTERNAL           LSAME, DLAMCH, DISNAN
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC          ABS, MAX, MIN
*
*   ..
*   .. External Subroutines ..
*   EXTERNAL           XERBLA
*
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
*   INFO = 0
*
*   IF( LSAME( TYPE, 'G' ) ) THEN
*       ITYPE = 0
*   ELSE IF( LSAME( TYPE, 'L' ) ) THEN
*       ITYPE = 1
*   ELSE IF( LSAME( TYPE, 'U' ) ) THEN
*       ITYPE = 2
*   ELSE IF( LSAME( TYPE, 'H' ) ) THEN
*       ITYPE = 3

```

```

      ELSE IF( LSAME( TYPE, 'B' ) ) THEN
        ITYPE = 4
      ELSE IF( LSAME( TYPE, 'Q' ) ) THEN
        ITYPE = 5
      ELSE IF( LSAME( TYPE, 'Z' ) ) THEN
        ITYPE = 6
      ELSE
        ITYPE = -1
      END IF
*
      IF( ITYPE.EQ.-1 ) THEN
        INFO = -1
      ELSE IF( CFROM.EQ.ZERO .OR. DISNAN(CFROM) ) THEN
        INFO = -4
      ELSE IF( DISNAN(CTO) ) THEN
        INFO = -5
      ELSE IF( M.LT.0 ) THEN
        INFO = -6
      ELSE IF( N.LT.0 .OR. ( ITYPE.EQ.4 .AND. N.NE.M ) .OR.
$          ( ITYPE.EQ.5 .AND. N.NE.M ) ) THEN
        INFO = -7
      ELSE IF( ITYPE.LE.3 .AND. LDA.LT.MAX( 1, M ) ) THEN
        INFO = -9
      ELSE IF( ITYPE.GE.4 ) THEN
        IF( KL.LT.0 .OR. KL.GT.MAX( M-1, 0 ) ) THEN
          INFO = -2
        ELSE IF( KU.LT.0 .OR. KU.GT.MAX( N-1, 0 ) .OR.
$          ( ( ITYPE.EQ.4 .OR. ITYPE.EQ.5 ) .AND. KL.NE.KU ) )
$          THEN
          INFO = -3
        ELSE IF( ( ITYPE.EQ.4 .AND. LDA.LT.KL+1 ) .OR.
$          ( ITYPE.EQ.5 .AND. LDA.LT.KU+1 ) .OR.
$          ( ITYPE.EQ.6 .AND. LDA.LT.2*KL+KU+1 ) ) THEN
          INFO = -9
        END IF
      END IF
*
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZLASCL', -INFO )
        RETURN
      END IF
*
*      Quick return if possible
*
      IF( N.EQ.0 .OR. M.EQ.0 )
$        RETURN
*
*      Get machine parameters
*
      SMLNUM = DLAMCH( 'S' )

```

```

        BIGNUM = ONE / SMLNUM
*
        CFROMC = CFROM
        CTOC = CT0
*
10  CONTINUE
    CFROM1 = CFROMC*SMLNUM
    IF( CFROM1.EQ.CFROMC ) THEN
!       CFROMC is an inf.  Multiply by a correctly signed zero for
!       finite CTOC, or a NaN if CTOC is infinite.
        MUL = CTOC / CFROMC
        DONE = .TRUE.
        CT01 = CTOC
    ELSE
        CT01 = CTOC / BIGNUM
        IF( CT01.EQ.CTOC ) THEN
!           CTOC is either 0 or an inf.  In both cases, CTOC itself
!           serves as the correct multiplication factor.
            MUL = CTOC
            DONE = .TRUE.
            CFROMC = ONE
        ELSE IF( ABS( CFROM1 ).GT.ABS( CTOC ) .AND. CTOC.NE.ZERO ) THEN
            MUL = SMLNUM
            DONE = .FALSE.
            CFROMC = CFROM1
        ELSE IF( ABS( CT01 ).GT.ABS( CFROMC ) ) THEN
            MUL = BIGNUM
            DONE = .FALSE.
            CTOC = CT01
        ELSE
            MUL = CTOC / CFROMC
            DONE = .TRUE.
        END IF
    END IF
*
    IF( ITYPE.EQ.0 ) THEN
*
*       Full matrix
*
        DO 30 J = 1, N
            DO 20 I = 1, M
                A( I, J ) = A( I, J )*MUL
            CONTINUE
        CONTINUE
*
    ELSE IF( ITYPE.EQ.1 ) THEN
*
*       Lower triangular matrix
*
        DO 50 J = 1, N

```

```

        DO 40 I = J, M
            A( I, J ) = A( I, J )*MUL
40      CONTINUE
50      CONTINUE
*
        ELSE IF( ITYPE.EQ.2 ) THEN
*
*          Upper triangular matrix
*
        DO 70 J = 1, N
            DO 60 I = 1, MIN( J, M )
                A( I, J ) = A( I, J )*MUL
60          CONTINUE
70        CONTINUE
*
        ELSE IF( ITYPE.EQ.3 ) THEN
*
*          Upper Hessenberg matrix
*
        DO 90 J = 1, N
            DO 80 I = 1, MIN( J+1, M )
                A( I, J ) = A( I, J )*MUL
80          CONTINUE
90        CONTINUE
*
        ELSE IF( ITYPE.EQ.4 ) THEN
*
*          Lower half of a symmetric band matrix
*
        K3 = KL + 1
        K4 = N + 1
        DO 110 J = 1, N
            DO 100 I = 1, MIN( K3, K4-J )
                A( I, J ) = A( I, J )*MUL
100          CONTINUE
110        CONTINUE
*
        ELSE IF( ITYPE.EQ.5 ) THEN
*
*          Upper half of a symmetric band matrix
*
        K1 = KU + 2
        K3 = KU + 1
        DO 130 J = 1, N
            DO 120 I = MAX( K1-J, 1 ), K3
                A( I, J ) = A( I, J )*MUL
120          CONTINUE
130        CONTINUE
*
        ELSE IF( ITYPE.EQ.6 ) THEN

```

```

*
*      Band matrix
*
      K1 = KL + KU + 2
      K2 = KL + 1
      K3 = 2*KL + KU + 1
      K4 = KL + KU + 1 + M
      DO 150 J = 1, N
        DO 140 I = MAX( K1-J, K2 ), MIN( K3, K4-J )
          A( I, J ) = A( I, J )*MUL
140      CONTINUE
150      CONTINUE
*
      END IF
*
      IF( .NOT.DONE )
        $ GO TO 10
*
      RETURN
*
*      End of ZLASCL
*
      END

```

— LAPACK zlascl —

```

(let* ((zero 0.0d0) (one 1.0d0))
  (declare (type (double-float 0.0d0 0.0d0) zero)
    (type (double-float 1.0d0 1.0d0) one) (ignorable zero one))
  (defun zlascl (type kl ku cfrom cto m n a lda info)
    (declare (type (simple-array character (*)) type)
      (type (f2cl-lib:integer4) info lda n m ku kl)
      (type (double-float) cto cfrom) (type (array f2cl-lib:complex16 (*)) a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
        (type character type-%data% type-%offset%))
      (prog
        ((bignum 0.0d0) (cfrom1 0.0d0) (cfromc 0.0d0) (cto1 0.0d0) (ctoc 0.0d0)
          (mul 0.0d0) (smlnum 0.0d0) (i 0) (itype 0) (j 0) (k1 0)
          (k2 0) (k3 0) (k4 0)
          (done nil))
        (declare (type (double-float) smlnum mul ctoc cto1
          cfromc cfrom1 bignum)
          (type (f2cl-lib:integer4) k4 k3 k2 k1 j itype i)
          (type f2cl-lib:logical done))
        (setf info 0)

```

```

(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "G")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 0))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "L")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 1))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "U")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 2))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "H")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 3))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "B")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 4))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "Q")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 5))
  ((multiple-value-bind (ret-val var-0 var-1) (lsame type "Z")
    (declare (ignore var-1)) (when var-0 (setf type var-0)) ret-val)
    (setf itype 6))
  (t (setf itype -1)))
(cond ((= itype (f2cl-lib:int-sub 1)) (setf info -1))
  ((or (= cfrom zero)
    (multiple-value-bind (ret-val var-0)
      (disnan cfrom) (declare (ignore))
      (when var-0 (setf cfrom var-0)) ret-val))
    (setf info -4))
  ((multiple-value-bind (ret-val var-0) (disnan cto) (declare (ignore))
    (when var-0 (setf cto var-0)) ret-val)
    (setf info -5))
  ((< m 0) (setf info -6))
  ((or (< n 0) (and (= itype 4) (/= n m)) (and (= itype 5) (/= n m)))
    (setf info -7))
  ((and (<= itype 3)
    (< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m))))
    (setf info -9))
  ((>= itype 4)
    (cond
      ((or (< kl 0)
        (> kl
          (max (the f2cl-lib:integer4
            (f2cl-lib:int-add m (f2cl-lib:int-sub 1)))
            (the f2cl-lib:integer4 0))))
        (setf info -2))
      ((or (< ku 0)
        (> ku
          (max (the f2cl-lib:integer4
            (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
            (the f2cl-lib:integer4 0))))
        (setf info -2))
      (t (setf info -2)))))

```

```

        (the f2cl-lib:integer4 0)))
    (and (or (= itype 4) (= itype 5)) (/= kl ku)))
    (setf info -3))
  ((or (and (= itype 4) (< lda (f2cl-lib:int-add kl 1)))
    (and (= itype 5) (< lda (f2cl-lib:int-add ku 1)))
    (and (= itype 6)
      (< lda (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))))
    (setf info -9))))
  (cond ((/= info 0)
    (xerbla "ZLASCL" (f2cl-lib:int-sub info)) (go end_label)))
  (if (or (= n 0) (= m 0)) (go end_label)) (setf smlnum (dlamch "S"))
  (setf bignum (/ one smlnum))
  (setf cfromc cfrom)
  (setf ctoc ctoc) label10
  (setf cfrom1 (* cfromc smlnum))
  (cond
    ((= cfrom1 cfromc)
      (setf mul (/ ctoc cfromc))
      (setf done f2cl-lib:%true%)
      (setf ctoc1 ctoc))
    (t (setf ctoc1 (/ ctoc bignum))
      (cond
        ((= ctoc1 ctoc) (setf mul ctoc) (setf done f2cl-lib:%true%)
          (setf cfromc one))
        ((and (> (abs cfrom1) (abs ctoc)) (/= ctoc zero)) (setf mul smlnum)
          (setf done f2cl-lib:%false%) (setf cfromc cfrom1))
        ((> (abs ctoc1) (abs cfromc))
          (setf mul bignum)
          (setf done f2cl-lib:%false%)
          (setf ctoc ctoc1))
        (t (setf mul (/ ctoc cfromc)) (setf done f2cl-lib:%true%))))))
  (cond
    ((= itype 0)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data% (i j)
                ((1 lda) (1 *)) a-%offset%))
                (* (f2cl-lib:fref a-%data% (i j)
                  ((1 lda) (1 *)) a-%offset%) mul))
              label20))
          label30)))
    ((= itype 1)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (f2cl-lib:fdo (i j (f2cl-lib:int-add i 1))
            ((> i m) nil)
            (tagbody
              (setf (f2cl-lib:fref a-%data% (i j)
                ((1 lda) (1 *)) a-%offset%))
                (* (f2cl-lib:fref a-%data% (i j)
                  ((1 lda) (1 *)) a-%offset%) mul))
              label20))
          label30)))
  )

```



```

(> i m) nil)
  (tagbody
    (setf (f2cl-lib:fref a-%data% (i j)
      ((1 lda) (1 *)) a-%offset%)
      (* (f2cl-lib:fref a-%data% (i j)
        ((1 lda) (1 *)) a-%offset%) mul))
      label140))
    label150)))
(= itype 2)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (min (the f2cl-lib:integer4 j)
          (the f2cl-lib:integer4 m)))
        nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%)
          (* (f2cl-lib:fref a-%data% (i j)
            ((1 lda) (1 *)) a-%offset%) mul))
          label160))
        label170)))
(= itype 3)
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (min (the f2cl-lib:integer4 (f2cl-lib:int-add j 1))
          (the f2cl-lib:integer4 m)))
        nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%)
          (* (f2cl-lib:fref a-%data% (i j)
            ((1 lda) (1 *)) a-%offset%) mul))
          label180))
        label190)))
(= itype 4) (setf k3 (f2cl-lib:int-add kl 1))
(setf k4 (f2cl-lib:int-add n 1))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j n) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i
        (min (the f2cl-lib:integer4 k3)
          (the f2cl-lib:integer4
            (f2cl-lib:int-add k4 (f2cl-lib:int-sub j))))))

```

```

        nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%)
          (* (f2cl-lib:fref a-%data% (i j)
            ((1 lda) (1 *)) a-%offset%) mul))
          label110))
      label110)))
    ((= itype 5) (setf k1 (f2cl-lib:int-add ku 2))
      (setf k3 (f2cl-lib:int-add ku 1))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i
          (max (the f2cl-lib:integer4
            (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
            (the f2cl-lib:integer4 1))
          (f2cl-lib:int-add i 1))
          (> i k3) nil)
        (tagbody
          (setf (f2cl-lib:fref a-%data% (i j)
            ((1 lda) (1 *)) a-%offset%)
            (* (f2cl-lib:fref a-%data% (i j)
              ((1 lda) (1 *)) a-%offset%) mul))
            label1120))
          label1130)))
    ((= itype 6) (setf k1 (f2cl-lib:int-add kl ku 2))
      (setf k2 (f2cl-lib:int-add kl 1))
      (setf k3 (f2cl-lib:int-add (f2cl-lib:int-mul 2 kl) ku 1))
      (setf k4 (f2cl-lib:int-add kl ku 1 m))
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i
          (max (the f2cl-lib:integer4
            (f2cl-lib:int-add k1 (f2cl-lib:int-sub j)))
            (the f2cl-lib:integer4 k2))
          (f2cl-lib:int-add i 1))
          (> i
            (min (the f2cl-lib:integer4 k3)
              (the f2cl-lib:integer4
                (f2cl-lib:int-add k4 (f2cl-lib:int-sub j))))))
          nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%)
          (* (f2cl-lib:fref a-%data% (i j) ((1 lda) (1 *))
            a-%offset%) mul))
          label1140))
        label1150))))

```

```
(if (not done) (go label10)) (go end_label) end_label
(return (values type nil nil cfrom cto nil nil nil nil info))))))
```

zlaset LAPACK

— zlaset.input —

```
)set break resume
)sys rm -f zlaset.output
)spool zlaset.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)
```

— zlaset.help —

```
=====
zlaset examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZLASET( UPLO, M, N, ALPHA, BETA, A, LDA )
```

```
.. Scalar Arguments ..
CHARACTER          UPLO
INTEGER            LDA, M, N
COMPLEX*16         ALPHA, BETA
..
.. Array Arguments ..
COMPLEX*16         A( LDA, * )
```

..

Purpose:

=====

ZLASET initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals.

Arguments:

=====

[in] UPLO

UPLO is CHARACTER*1

Specifies the part of the matrix A to be set.

= 'U': Upper triangular part is set. The lower triangle is unchanged.

= 'L': Lower triangular part is set. The upper triangle is unchanged.

Otherwise: All of the matrix A is set.

[in] M

M is INTEGER

On entry, M specifies the number of rows of A.

[in] N

N is INTEGER

On entry, N specifies the number of columns of A.

[in] ALPHA

ALPHA is COMPLEX*16

All the offdiagonal array elements are set to ALPHA.

[in] BETA

BETA is COMPLEX*16

All the diagonal array elements are set to BETA.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)

On entry, the m by n matrix A.

On exit, $A(i,j) = \text{ALPHA}$, $1 \leq i \leq m$, $1 \leq j \leq n$, $i \neq j$; $A(i,i) = \text{BETA}$, $1 \leq i \leq \min(m,n)$

[in] LDA

LDA is INTEGER
 The leading dimension of the array A. LDA \geq max(1,M).

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zlaset.f —

```
* =====
*      SUBROUTINE ZLASET( UPLO, M, N, ALPHA, BETA, A, LDA )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER            UPLO
*      INTEGER              LDA, M, N
*      COMPLEX*16           ALPHA, BETA
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16           A( LDA, * )
*
*      ..
*
*      =====
*
*      .. Local Scalars ..
*      INTEGER              I, J
*
*      ..
*      .. External Functions ..
*      LOGICAL              LSAME
*      EXTERNAL             LSAME
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC            MIN
*
*      ..
*      .. Executable Statements ..
```

```

*
      IF( LSAME( UPLO, 'U' ) ) THEN
*
*       Set the diagonal to BETA and the strictly upper triangular
*       part of the array to ALPHA.
*
      DO 20 J = 2, N
        DO 10 I = 1, MIN( J-1, M )
          A( I, J ) = ALPHA
10      CONTINUE
20      CONTINUE
      DO 30 I = 1, MIN( N, M )
        A( I, I ) = BETA
30      CONTINUE
*
      ELSE IF( LSAME( UPLO, 'L' ) ) THEN
*
*       Set the diagonal to BETA and the strictly lower triangular
*       part of the array to ALPHA.
*
      DO 50 J = 1, MIN( M, N )
        DO 40 I = J + 1, M
          A( I, J ) = ALPHA
40      CONTINUE
50      CONTINUE
      DO 60 I = 1, MIN( N, M )
        A( I, I ) = BETA
60      CONTINUE
*
      ELSE
*
*       Set the array to BETA on the diagonal and ALPHA on the
*       offdiagonal.
*
      DO 80 J = 1, N
        DO 70 I = 1, M
          A( I, J ) = ALPHA
70      CONTINUE
80      CONTINUE
      DO 90 I = 1, MIN( M, N )
        A( I, I ) = BETA
90      CONTINUE
      END IF
*
      RETURN
*
*     End of ZLASET
*
      END

```

— LAPACK zlaset —

```

(defun zlaset (uplo m n alpha beta a lda)
  (declare (type (simple-array character (*)) uplo)
    (type (f2cl-lib:integer4) lda n m) (type (f2cl-lib:complex16) beta alpha)
    (type (array f2cl-lib:complex16 (*)) a))
  (f2cl-lib:with-multi-array-data
    ((a f2cl-lib:complex16 a-%data% a-%offset%)
     (uplo character uplo-%data% uplo-%offset%))
    (prog ((i 0) (j 0))
      (declare (type (f2cl-lib:integer4) j i))
      (cond
        ((multiple-value-bind (ret-val var-0 var-1) (lsame uplo "U")
          (declare (ignore var-1)) (when var-0 (setf uplo var-0)) ret-val)
         (f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
           (> j n) nil)
         (tagbody
           (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
             (> i
              (min (the f2cl-lib:integer4
                        (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
                   (the f2cl-lib:integer4 m)))
              nil)
           (tagbody
             (setf (f2cl-lib:fref a-%data% (i j)
                                   ((1 lda) (1 *)) a-%offset%) alpha)
               label10))
           label20))
         (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
           (> i
            (min (the f2cl-lib:integer4 n)
                  (the f2cl-lib:integer4 m)))
            nil)
         (tagbody
           (setf (f2cl-lib:fref a-%data% (i i)
                                   ((1 lda) (1 *)) a-%offset%) beta)
               label30)))
        ((multiple-value-bind (ret-val var-0 var-1) (lsame uplo "L")
          (declare (ignore var-1)) (when var-0 (setf uplo var-0)) ret-val)
         (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
           (> j
            (min (the f2cl-lib:integer4 m)
                  (the f2cl-lib:integer4 n)))
            nil)
         (tagbody
           (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                           (f2cl-lib:int-add i 1))

```

```

        (> i m)
        nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%) alpha)
          label40))
        label50))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i
          (min (the f2cl-lib:integer4 n)
            (the f2cl-lib:integer4 m)))
          nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i i)
          ((1 lda) (1 *)) a-%offset%) beta)
          label60)))
    (t
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i m) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data% (i j)
              ((1 lda) (1 *)) a-%offset%) alpha)
              label70))
            label80))
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i
            (min (the f2cl-lib:integer4 m)
              (the f2cl-lib:integer4 n)))
            nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data% (i i)
              ((1 lda) (1 *)) a-%offset%) beta)
              label90))))
      (go end_label)
    end_label
    (return (values uplo nil nil nil nil nil nil)))
  ))

```

zlassq LAPACK

— zlassq.input —


```

)set break resume
)sys rm -f zlassq.output
)spool zlassq.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlassq.help —

```

=====
zlassq examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```

SUBROUTINE ZLASSQ( N, X, INCX, SCALE, SUMSQ )

```

```

.. Scalar Arguments ..
INTEGER          INCX, N
DOUBLE PRECISION SCALE, SUMSQ
..
.. Array Arguments ..
COMPLEX*16       X( * )
..

```

Purpose:
 =====

ZLASSQ returns the values scl and ssq such that

$$(scl**2)*ssq = x(1)**2 + \dots + x(n)**2 + (scale**2)*sumsq,$$

where $x(i) = \text{abs}(X(1 + (i - 1)*INCX))$. The value of sumsq is assumed to be at least unity and the value of ssq will then satisfy

```
1.0 .le. ssq .le. ( sumsq + 2*n ).
```

scale is assumed to be non-negative and scl returns the value

```
scl = max( scale, abs( real( x( i ) ) ), abs( aimag( x( i ) ) ) ),
          i
```

scale and sumsq must be supplied in SCALE and SUMSQ respectively.
SCALE and SUMSQ are overwritten by scl and ssq respectively.

The routine makes only one pass through the vector X.

Arguments:

=====

[in] N

N is INTEGER

The number of elements to be used from the vector X.

[in] X

X is COMPLEX*16 array, dimension (N)

The vector x as described above.

$x(i) = X(1 + (i - 1) * INCX)$, $1 \leq i \leq n$.

[in] INCX

INCX is INTEGER

The increment between successive values of the vector X.

INCX > 0.

[in,out] SCALE

SCALE is DOUBLE PRECISION

On entry, the value scale in the equation above.

On exit, SCALE is overwritten with the value scl.

[in,out] SUMSQ

SUMSQ is DOUBLE PRECISION

On entry, the value sumsq in the equation above.

On exit, SUMSQ is overwritten with the value ssq.

Authors:

=====

Univ. of Tennessee

Univ. of California Berkeley

Univ. of Colorado Denver

NAG Ltd.

November 2011

— zlassq.f —

```

* =====
*      SUBROUTINE ZLASSQ( N, X, INCX, SCALE, SUMSQ )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          INCX, N
*      DOUBLE PRECISION  SCALE, SUMSQ
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       X( * )
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION  ZERO
*      PARAMETER         ( ZERO = 0.0D+0 )
*
*      ..
*      .. Local Scalars ..
*      INTEGER          IX
*      DOUBLE PRECISION  TEMP1
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC        ABS, DBLE, DIMAG
*
*      ..
*      .. Executable Statements ..
*
*      IF( N.GT.0 ) THEN
*        DO 10 IX = 1, 1 + ( N-1 )*INCX, INCX
*          IF( DBLE( X( IX ) ).NE.ZERO ) THEN
*            TEMP1 = ABS( DBLE( X( IX ) ) )
*            IF( SCALE.LT.TEMP1 ) THEN
*              SUMSQ = 1 + SUMSQ*( SCALE / TEMP1 )**2
*              SCALE = TEMP1
*            ELSE
*              SUMSQ = SUMSQ + ( TEMP1 / SCALE )**2

```

```

        END IF
      END IF
      IF( DIMAG( X( IX ) ).NE.ZERO ) THEN
        TEMP1 = ABS( DIMAG( X( IX ) ) )
        IF( SCALE.LT.TEMP1 ) THEN
          SUMSQ = 1 + SUMSQ*( SCALE / TEMP1 )**2
          SCALE = TEMP1
        ELSE
          SUMSQ = SUMSQ + ( TEMP1 / SCALE )**2
        END IF
      END IF
10    CONTINUE
      END IF
*
      RETURN
*
*    End of ZLASSQ
*
      END

```

— LAPACK zlassq —

```

(let* ((zero 0.0))
  (declare (type (double-float 0.0 0.0) zero))
  (defun zlassq (n x incx scale sumsq)
    (declare (type (double-float) sumsq scale)
      (type (simple-array (complex double-float) (*)) x)
      (type fixnum incx n))
    (f2cl-lib:with-multi-array-data
      ((x (complex double-float) x-%data% x-%offset%))
      (prog ((temp1 0.0) (ix 0))
        (declare (type (double-float) temp1) (type fixnum ix))
        (cond
          ((> n 0)
            (f2cl-lib:fdo (ix 1 (f2cl-lib:int-add ix incx))
              ((> ix
                (f2cl-lib:int-add 1
                  (f2cl-lib:int-mul
                    (f2cl-lib:int-add n
                      (f2cl-lib:int-sub 1))
                    incx))))
              nil)
            (tagbody
              (cond
                ((/= (coerce (realpart
                  (f2cl-lib:fref x (ix) ((1 *)))) 'double-float) zero)

```

```

      (setf temp1
        (abs
          (coerce (realpart
                    (f2cl-lib:fref x-%data% (ix)
                      ((1 *)) x-%offset%)) 'double-float)))
      (cond
        ((< scale temp1)
         (setf sumsq (+ 1 (* sumsq (expt (/ scale temp1) 2))))
         (setf scale temp1))
        (t
         (setf sumsq (+ sumsq (expt (/ temp1 scale) 2))))))
      (cond
        ((/= (f2cl-lib:dimag (f2cl-lib:fref x (ix) ((1 *))) zero)
              (setf temp1
                (abs
                  (f2cl-lib:dimag
                    (f2cl-lib:fref x-%data% (ix) ((1 *)) x-%offset%))))
              (cond
                ((< scale temp1)
                 (setf sumsq (+ 1 (* sumsq (expt (/ scale temp1) 2))))
                 (setf scale temp1))
                (t
                 (setf sumsq (+ sumsq (expt (/ temp1 scale) 2))))))))
        (return (values nil nil nil scale sumsq))))

```

zlatrs LAPACK

— zlatrs.input —

```

)set break resume
)sys rm -f zlatrs.output
)spool zlatrs.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zlatrs.help —

=====

zlatrs examples

Man Page Details

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
SUBROUTINE ZLATRS( UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE,
                  CNORM, INFO )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          DIAG, NORMIN, TRANS, UPLO
```

```
INTEGER            INFO, LDA, N
```

```
DOUBLE PRECISION   SCALE
```

```
..
```

```
.. Array Arguments ..
```

```
DOUBLE PRECISION   CNORM( * )
```

```
COMPLEX*16         A( LDA, * ), X( * )
```

```
..
```

Purpose:

ZLATRS solves one of the triangular systems

$$A * x = s*b, \quad A^{*T} * x = s*b, \quad \text{or} \quad A^{*H} * x = s*b,$$

with scaling to prevent overflow. Here A is an upper or lower triangular matrix, A**T denotes the transpose of A, A**H denotes the conjugate transpose of A, x and b are n-element vectors, and s is a scaling factor, usually less than or equal to 1, chosen so that the components of x will be less than the overflow threshold. If the unscaled problem will not cause overflow, the Level 2 BLAS routine ZTRSV is called. If the matrix A is singular ($A(j,j) = 0$ for some j), then s is set to 0 and a non-trivial solution to $A*x = 0$ is returned.

Arguments:

[in] UPLO

UPLO is CHARACTER*1

Specifies whether the matrix A is upper or lower triangular.

= 'U': Upper triangular
 = 'L': Lower triangular

[in] TRANS

TRANS is CHARACTER*1
 Specifies the operation applied to A.
 = 'N': Solve $A * x = s*b$ (No transpose)
 = 'T': Solve $A**T * x = s*b$ (Transpose)
 = 'C': Solve $A**H * x = s*b$ (Conjugate transpose)

[in] DIAG

DIAG is CHARACTER*1
 Specifies whether or not the matrix A is unit triangular.
 = 'N': Non-unit triangular
 = 'U': Unit triangular

[in] NORMIN

NORMIN is CHARACTER*1
 Specifies whether CNORM has been set or not.
 = 'Y': CNORM contains the column norms on entry
 = 'N': CNORM is not set on entry. On exit, the norms will
 be computed and stored in CNORM.

[in] N

N is INTEGER
 The order of the matrix A. $N \geq 0$.

[in] A

A is COMPLEX*16 array, dimension (LDA,N)
 The triangular matrix A. If UPLO = 'U', the leading n by n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n by n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

[in] LDA

LDA is INTEGER
 The leading dimension of the array A. $LDA \geq \max(1, N)$.

[in,out] X

X is COMPLEX*16 array, dimension (N)
 On entry, the right hand side b of the triangular system.
 On exit, X is overwritten by the solution vector x.

[out] SCALE

SCALE is DOUBLE PRECISION
 The scaling factor s for the triangular system
 $A * x = s*b$, $A**T * x = s*b$, or $A**H * x = s*b$.
 If SCALE = 0, the matrix A is singular or badly scaled, and
 the vector x is an exact or approximate solution to $A*x = 0$.

[in,out] CNORM

CNORM is or output) DOUBLE PRECISION array, dimension (N)

 If NORMIN = 'Y', CNORM is an input argument and CNORM(j)
 contains the norm of the off-diagonal part of the j-th column
 of A. If TRANS = 'N', CNORM(j) must be greater than or equal
 to the infinity-norm, and if TRANS = 'T' or 'C', CNORM(j)
 must be greater than or equal to the 1-norm.

 If NORMIN = 'N', CNORM is an output argument and CNORM(j)
 returns the 1-norm of the offdiagonal part of the j-th column
 of A.

[out] INFO

INFO is INTEGER
 = 0: successful exit
 < 0: if INFO = -k, the k-th argument had an illegal value

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:
 =====

A rough bound on x is computed; if that is less than overflow, ZTRSV
 is called, otherwise, specific code is used which checks for possible
 overflow or divide-by-zero at every operation.

A columnwise scheme is used for solving $A*x = b$. The basic algorithm

if A is lower triangular is

```

x[1:n] := b[1:n]
for j = 1, ..., n
    x(j) := x(j) / A(j,j)
    x[j+1:n] := x[j+1:n] - x(j) * A[j+1:n,j]
end

```

Define bounds on the components of x after j iterations of the loop:

$M(j)$ = bound on $x[1:j]$

$G(j)$ = bound on $x[j+1:n]$

Initially, let $M(0) = 0$ and $G(0) = \max\{x(i), i=1, \dots, n\}$.

Then for iteration j+1 we have

$M(j+1) \leq G(j) / |A(j+1,j+1)|$

$G(j+1) \leq G(j) + M(j+1) * |A[j+2:n,j+1]|$

$\leq G(j) (1 + CNORM(j+1) / |A(j+1,j+1)|)$

where $CNORM(j+1)$ is greater than or equal to the infinity-norm of column j+1 of A, not counting the diagonal. Hence

$G(j) \leq G(0) \prod_{1 \leq i \leq j} (1 + CNORM(i) / |A(i,i)|)$

and

$|x(j)| \leq (G(0) / |A(j,j)|) \prod_{1 \leq i < j} (1 + CNORM(i) / |A(i,i)|)$

Since $|x(j)| \leq M(j)$, we use the Level 2 BLAS routine ZTRSV if the reciprocal of the largest $M(j)$, $j=1, \dots, n$, is larger than $\max(\text{underflow}, 1/\text{overflow})$.

The bound on $x(j)$ is also used to determine when a step in the columnwise method can be performed without fear of overflow. If the computed bound is greater than a large constant, x is scaled to prevent overflow, but if the bound overflows, x is set to 0, $x(j)$ to 1, and scale to 0, and a non-trivial solution to $A*x = 0$ is found.

Similarly, a row-wise scheme is used to solve $A^{**T} * x = b$ or $A^{**H} * x = b$. The basic algorithm for A upper triangular is

```

for j = 1, ..., n
    x(j) := ( b(j) - A[1:j-1,j]' * x[1:j-1] ) / A(j,j)
end

```

We simultaneously compute two bounds

$G(j)$ = bound on $(b(i) - A[1:i-1,i]' * x[1:i-1])$, $1 \leq i \leq j$

$M(j)$ = bound on $x(i)$, $1 \leq i \leq j$

The initial values are $G(0) = 0$, $M(0) = \max\{b(i), i=1, \dots, n\}$, and we

add the constraint $G(j) \geq G(j-1)$ and $M(j) \geq M(j-1)$ for $j \geq 1$.
Then the bound on $x(j)$ is

$$M(j) \leq M(j-1) * (1 + CNORM(j)) / | A(j,j) |$$

$$\leq M(0) * \text{product} ((1 + CNORM(i)) / |A(i,i)|)$$

$$1 \leq i \leq j$$

and we can safely call ZTRSV if $1/M(n)$ and $1/G(n)$ are both greater than $\max(\text{underflow}, 1/\text{overflow})$.

— zlatrs.f —

```
* =====
*      SUBROUTINE ZLATRS( UPLO, TRANS, DIAG, NORMIN, N, A, LDA, X, SCALE,
*      $                  CNORM, INFO )
*
*      -- LAPACK auxiliary routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          DIAG, NORMIN, TRANS, UPLO
*      INTEGER            INFO, LDA, N
*      DOUBLE PRECISION    SCALE
*
*      ..
*      .. Array Arguments ..
*      DOUBLE PRECISION    CNORM( * )
*      COMPLEX*16          A( LDA, * ), X( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
*      DOUBLE PRECISION    ZERO, HALF, ONE, TWO
*      PARAMETER            ( ZERO = 0.0D+0, HALF = 0.5D+0, ONE = 1.0D+0,
*      $                    TWO = 2.0D+0 )
*
*      ..
*      .. Local Scalars ..
*      LOGICAL             NOTRAN, NOUNIT, UPPER
*      INTEGER             I, IMAX, J, JFIRST, JINC, JLAST
*      DOUBLE PRECISION    BIGNUM, GROW, REC, SMLNUM, TJJ, TMAX, TSCAL,
*      $                    XBND, XJ, XMAX
*      COMPLEX*16          CSUMJ, TJJS, USCAL, ZDUM
*
*      ..
```

```

*      .. External Functions ..
LOGICAL          LSAME
INTEGER          IDAMAX, IZAMAX
DOUBLE PRECISION DLAMCH, DZASUM
COMPLEX*16       ZDOTC, ZDOTU, ZLADIV
EXTERNAL         LSAME, IDAMAX, IZAMAX, DLAMCH, DZASUM, ZDOTC,
$               ZDOTU, ZLADIV
*
*      ..
*      .. External Subroutines ..
EXTERNAL         DSCAL, XERBLA, ZAXPY, ZDSCAL, ZTRSV
*
*      ..
*      .. Intrinsic Functions ..
INTRINSIC        ABS, DBLE, DCMPLX, DCONJG, DIMAG, MAX, MIN
*
*      ..
*      .. Statement Functions ..
DOUBLE PRECISION CABS1, CABS2
*
*      ..
*      .. Statement Function definitions ..
CABS1( ZDUM ) = ABS( DBLE( ZDUM ) ) + ABS( DIMAG( ZDUM ) )
CABS2( ZDUM ) = ABS( DBLE( ZDUM ) / 2.DO ) +
$             ABS( DIMAG( ZDUM ) / 2.DO )
*
*      ..
*      .. Executable Statements ..
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      NOTRAN = LSAME( TRANS, 'N' )
      NOUNIT = LSAME( DIAG, 'N' )
*
*      Test the input parameters.
*
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
          INFO = -1
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'T' ) .AND. .NOT.
$           LSAME( TRANS, 'C' ) ) THEN
          INFO = -2
      ELSE IF( .NOT.NOUNIT .AND. .NOT.LSAME( DIAG, 'U' ) ) THEN
          INFO = -3
      ELSE IF( .NOT.LSAME( NORMIN, 'Y' ) .AND. .NOT.
$           LSAME( NORMIN, 'N' ) ) THEN
          INFO = -4
      ELSE IF( N.LT.0 ) THEN
          INFO = -5
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
          INFO = -7
      END IF
      IF( INFO.NE.0 ) THEN
          CALL XERBLA( 'ZLATRS', -INFO )
          RETURN
      END IF

```

```

*
*   Quick return if possible
*
*   IF( N.EQ.0 )
*     $ RETURN
*
*   Determine machine dependent parameters to control overflow.
*
*   SMLNUM = DLAMCH( 'Safe minimum' )
*   BIGNUM = ONE / SMLNUM
*   CALL DLABAD( SMLNUM, BIGNUM )
*   SMLNUM = SMLNUM / DLAMCH( 'Precision' )
*   BIGNUM = ONE / SMLNUM
*   SCALE = ONE
*
*   IF( LSAME( NORMIN, 'N' ) ) THEN
*
*     Compute the 1-norm of each column, not including the diagonal.
*
*     IF( UPPER ) THEN
*
*       A is upper triangular.
*
*       DO 10 J = 1, N
*         CNORM( J ) = DZASUM( J-1, A( 1, J ), 1 )
10      CONTINUE
*     ELSE
*
*       A is lower triangular.
*
*       DO 20 J = 1, N - 1
*         CNORM( J ) = DZASUM( N-J, A( J+1, J ), 1 )
20      CONTINUE
*       CNORM( N ) = ZERO
*     END IF
*   END IF
*
*   Scale the column norms by TSCAL if the maximum element in CNORM is
*   greater than BIGNUM/2.
*
*   IMAX = IDAMAX( N, CNORM, 1 )
*   TMAX = CNORM( IMAX )
*   IF( TMAX.LE.BIGNUM*HALF ) THEN
*     TSCAL = ONE
*   ELSE
*     TSCAL = HALF / ( SMLNUM*TMAX )
*     CALL DSCAL( N, TSCAL, CNORM, 1 )
*   END IF
*
*   Compute a bound on the computed solution vector to see if the

```

```

*      Level 2 BLAS routine ZTRSV can be used.
*
      XMAX = ZERO
      DO 30 J = 1, N
          XMAX = MAX( XMAX, CABS2( X( J ) ) )
30 CONTINUE
      XBND = XMAX
*
      IF( NOTRAN ) THEN
*
*          Compute the growth in A * x = b.
*
          IF( UPPER ) THEN
              JFIRST = N
              JLAST = 1
              JINC = -1
          ELSE
              JFIRST = 1
              JLAST = N
              JINC = 1
          END IF
*
          IF( TSCAL.NE.ONE ) THEN
              GROW = ZERO
              GO TO 60
          END IF
*
          IF( NOUNIT ) THEN
*
*              A is non-unit triangular.
*
*              Compute GROW = 1/G(j) and XBND = 1/M(j).
*              Initially, G(0) = max{x(i), i=1,...,n}.
*
              GROW = HALF / MAX( XBND, SMLNUM )
              XBND = GROW
              DO 40 J = JFIRST, JLAST, JINC
*
*                  Exit the loop if the growth factor is too small.
*
*                  IF( GROW.LE.SMLNUM )
$                      GO TO 60
*
*                  TJJS = A( J, J )
*                  TJJ = CABS1( TJJS )
*
*                  IF( TJJ.GE.SMLNUM ) THEN
*
*                      M(j) = G(j-1) / abs(A(j,j))
*

```

```

        XBND = MIN( XBND, MIN( ONE, TJJ ) * GROW )
    ELSE
*
*         M(j) could overflow, set XBND to 0.
*
        XBND = ZERO
    END IF
*
    IF( TJJ + CNORM( J ) .GE. SMLNUM ) THEN
*
*         G(j) = G(j-1) * ( 1 + CNORM(j) / abs(A(j,j)) )
*
        GROW = GROW * ( TJJ / ( TJJ + CNORM( J ) ) )
    ELSE
*
*         G(j) could overflow, set GROW to 0.
*
        GROW = ZERO
    END IF
40    CONTINUE
        GROW = XBND
    ELSE
*
*         A is unit triangular.
*
*         Compute GROW = 1/G(j), where G(0) = max{x(i), i=1,...,n}.
*
        GROW = MIN( ONE, HALF / MAX( XBND, SMLNUM ) )
        DO 50 J = JFIRST, JLAST, JINC
*
*         Exit the loop if the growth factor is too small.
*
        IF( GROW.LE.SMLNUM )
            $          GO TO 60
*
*         G(j) = G(j-1) * ( 1 + CNORM(j) )
*
        GROW = GROW * ( ONE / ( ONE + CNORM( J ) ) )
50    CONTINUE
        END IF
60    CONTINUE
*
    ELSE
*
*         Compute the growth in A**T * x = b or A**H * x = b.
*
    IF( UPPER ) THEN
        JFIRST = 1
        JLAST = N
        JINC = 1

```

```

        ELSE
            JFIRST = N
            JLAST = 1
            JINC = -1
        END IF
*
        IF( TSCAL.NE.ONE ) THEN
            GROW = ZERO
            GO TO 90
        END IF
*
        IF( NOUNIT ) THEN
*
*           A is non-unit triangular.
*
*           Compute GROW = 1/G(j) and XBND = 1/M(j).
*           Initially, M(0) = max{x(i), i=1,...,n}.
*
            GROW = HALF / MAX( XBND, SMLNUM )
            XBND = GROW
            DO 70 J = JFIRST, JLAST, JINC
*
*           Exit the loop if the growth factor is too small.
*
*
                IF( GROW.LE.SMLNUM )
$                   GO TO 90
*
*           G(j) = max( G(j-1), M(j-1)*( 1 + CNORM(j) ) )
*
                G(j) = max( G(j-1), M(j-1)*( 1 + CNORM(j) ) )
*
                XJ = ONE + CNORM( J )
                GROW = MIN( GROW, XBND / XJ )
*
                TJJS = A( J, J )
                TJJ = CABS1( TJJS )
*
                IF( TJJ.GE.SMLNUM ) THEN
*
*                   M(j) = M(j-1)*( 1 + CNORM(j) ) / abs(A(j,j))
*
                    M(j) = M(j-1)*( 1 + CNORM(j) ) / abs(A(j,j))
*
                    IF( XJ.GT.TJJ )
$                        XBND = XBND*( TJJ / XJ )
                    ELSE
*
*                   M(j) could overflow, set XBND to 0.
*
                        XBND = ZERO
                    END IF
20                CONTINUE
                GROW = MIN( GROW, XBND )
            ELSE

```

```

*
*      A is unit triangular.
*
*      Compute GROW = 1/G(j), where G(0) = max{x(i), i=1,...,n}.
*
      GROW = MIN( ONE, HALF / MAX( XBND, SMLNUM ) )
      DO 80 J = JFIRST, JLAST, JINC
*
*          Exit the loop if the growth factor is too small.
*
*          IF( GROW.LE.SMLNUM )
$              GO TO 90
*
*          G(j) = ( 1 + CNORM(j) ) * G(j-1)
*
*          XJ = ONE + CNORM( J )
*          GROW = GROW / XJ
80      CONTINUE
      END IF
90      CONTINUE
      END IF
*
      IF( ( GROW*TSCAL ).GT.SMLNUM ) THEN
*
*          Use the Level 2 BLAS solve if the reciprocal of the bound on
*          elements of X is not too small.
*
          CALL ZTRSV( UPLO, TRANS, DIAG, N, A, LDA, X, 1 )
      ELSE
*
*          Use a Level 1 BLAS solve, scaling intermediate results.
*
          IF( XMAX.GT.BIGNUM*HALF ) THEN
*
*          Scale X so that its components are less than or equal to
*          BIGNUM in absolute value.
*
          SCALE = ( BIGNUM*HALF ) / XMAX
          CALL ZDSCAL( N, SCALE, X, 1 )
          XMAX = BIGNUM
      ELSE
          XMAX = XMAX*TWO
      END IF
*
      IF( NOTRAN ) THEN
*
*          Solve A * x = b
*
          DO 120 J = JFIRST, JLAST, JINC
*

```



```

*          Compute x(j) = b(j) / A(j,j), scaling x if necessary.
*
      XJ = CABS1( X( J ) )
      IF( NOUNIT ) THEN
        TJJS = A( J, J )*TSCAL
      ELSE
        TJJS = TSCAL
        IF( TSCAL.EQ.ONE )
$          GO TO 110
      END IF
      TJJ = CABS1( TJJS )
      IF( TJJ.GT.SMLNUM ) THEN
*
*          abs(A(j,j)) > SMLNUM:
*
      IF( TJJ.LT.ONE ) THEN
        IF( XJ.GT.TJJ*BIGNUM ) THEN
*
*          Scale x by 1/b(j).
*
          REC = ONE / XJ
          CALL ZDSCAL( N, REC, X, 1 )
          SCALE = SCALE*REC
          XMAX = XMAX*REC
        END IF
      END IF
      X( J ) = ZLADIV( X( J ), TJJS )
      XJ = CABS1( X( J ) )
      ELSE IF( TJJ.GT.ZERO ) THEN
*
*          0 < abs(A(j,j)) <= SMLNUM:
*
      IF( XJ.GT.TJJ*BIGNUM ) THEN
*
*          Scale x by (1/abs(x(j)))*abs(A(j,j))*BIGNUM
*          to avoid overflow when dividing by A(j,j).
*
          REC = ( TJJ*BIGNUM ) / XJ
          IF( CNORM( J ).GT.ONE ) THEN
*
*          Scale by 1/CNORM(j) to avoid overflow when
*          multiplying x(j) times column j.
*
          REC = REC / CNORM( J )
        END IF
      END IF
      CALL ZDSCAL( N, REC, X, 1 )
      SCALE = SCALE*REC
      XMAX = XMAX*REC
    END IF
    X( J ) = ZLADIV( X( J ), TJJS )

```

```

XJ = CABS1( X( J ) )
ELSE
*
*      A(j,j) = 0: Set x(1:n) = 0, x(j) = 1, and
*      scale = 0, and compute a solution to A*x = 0.
*
      DO 100 I = 1, N
        X( I ) = ZERO
100    CONTINUE
      X( J ) = ONE
      XJ = ONE
      SCALE = ZERO
      XMAX = ZERO
      END IF
110    CONTINUE
*
*      Scale x if necessary to avoid overflow when adding a
*      multiple of column j of A.
*
      IF( XJ.GT.ONE ) THEN
        REC = ONE / XJ
        IF( CNORM( J ).GT.( BIGNUM-XMAX )*REC ) THEN
*
*          Scale x by 1/(2*abs(x(j))).
*
          REC = REC*HALF
          CALL ZDSCAL( N, REC, X, 1 )
          SCALE = SCALE*REC
        END IF
      ELSE IF( XJ*CNORM( J ).GT.( BIGNUM-XMAX ) ) THEN
*
*          Scale x by 1/2.
*
        CALL ZDSCAL( N, HALF, X, 1 )
        SCALE = SCALE*HALF
      END IF
*
      IF( UPPER ) THEN
        IF( J.GT.1 ) THEN
*
*          Compute the update
*          x(1:j-1) := x(1:j-1) - x(j) * A(1:j-1,j)
*
          CALL ZAXPY( J-1, -X( J )*TSCAL, A( 1, J ), 1, X,
1          1 )
          I = IZAMAX( J-1, X, 1 )
          XMAX = CABS1( X( I ) )
        END IF
      ELSE
        IF( J.LT.N ) THEN

```

```

*
*           Compute the update
*           x(j+1:n) := x(j+1:n) - x(j) * A(j+1:n,j)
*
*           CALL ZAXPY( N-J, -X( J )*TSCAL, A( J+1, J ), 1,
$           X( J+1 ), 1 )
*           I = J + IZAMAX( N-J, X( J+1 ), 1 )
*           XMAX = CABS1( X( I ) )
*           END IF
*           END IF
120      CONTINUE
*
*      ELSE IF( LSAME( TRANS, 'T' ) ) THEN
*
*          Solve A**T * x = b
*
*          DO 170 J = JFIRST, JLAST, JINC
*
*              Compute x(j) = b(j) - sum A(k,j)*x(k).
*                      k<>j
*
*              XJ = CABS1( X( J ) )
*              USCAL = TSCAL
*              REC = ONE / MAX( XMAX, ONE )
*              IF( CNORM( J ).GT.( BIGNUM-XJ )*REC ) THEN
*
*                  If x(j) could overflow, scale x by 1/(2*XMAX).
*
*                  REC = REC*HALF
*                  IF( NOUNIT ) THEN
*                      TJJS = A( J, J )*TSCAL
*                  ELSE
*                      TJJS = TSCAL
*                  END IF
*                  TJJ = CABS1( TJJS )
*                  IF( TJJ.GT.ONE ) THEN
*
*                      Divide by A(j,j) when scaling x if A(j,j) > 1.
*
*                      REC = MIN( ONE, REC*TJJ )
*                      USCAL = ZLADIV( USCAL, TJJS )
*                  END IF
*                  IF( REC.LT.ONE ) THEN
*                      CALL ZDSCAL( N, REC, X, 1 )
*                      SCALE = SCALE*REC
*                      XMAX = XMAX*REC
*                  END IF
*              END IF
*
*          CSUMJ = ZERO

```

```

IF( USCAL.EQ.DCMPLX( ONE ) ) THEN
*
*      If the scaling needed for A in the dot product is 1,
*      call ZDOTU to perform the dot product.
*
      IF( UPPER ) THEN
        CSUMJ = ZDOTU( J-1, A( 1, J ), 1, X, 1 )
      ELSE IF( J.LT.N ) THEN
        CSUMJ = ZDOTU( N-J, A( J+1, J ), 1, X( J+1 ), 1 )
      END IF
    ELSE
*
*      Otherwise, use in-line code for the dot product.
*
      IF( UPPER ) THEN
        DO 130 I = 1, J - 1
          CSUMJ = CSUMJ + ( A( I, J )*USCAL )*X( I )
130        CONTINUE
      ELSE IF( J.LT.N ) THEN
        DO 140 I = J + 1, N
          CSUMJ = CSUMJ + ( A( I, J )*USCAL )*X( I )
140        CONTINUE
      END IF
    END IF
*
IF( USCAL.EQ.DCMPLX( TSCAL ) ) THEN
*
*      Compute x(j) := ( x(j) - CSUMJ ) / A(j,j) if 1/A(j,j)
*      was not used to scale the dotproduct.
*
      X( J ) = X( J ) - CSUMJ
      XJ = CABS1( X( J ) )
      IF( NOUNIT ) THEN
        TJJS = A( J, J )*TSCAL
      ELSE
        TJJS = TSCAL
        IF( TSCAL.EQ.ONE )
$          GO TO 160
        END IF
*
*      Compute x(j) = x(j) / A(j,j), scaling if necessary.
*
      TJJ = CABS1( TJJS )
      IF( TJJ.GT.SMLNUM ) THEN
*
*          abs(A(j,j)) > SMLNUM:
*
          IF( TJJ.LT.ONE ) THEN
            IF( XJ.GT.TJJ*BIGNUM ) THEN

```

```

*           Scale X by 1/abs(x(j)).
*
*           REC = ONE / XJ
*           CALL ZDSCAL( N, REC, X, 1 )
*           SCALE = SCALE*REC
*           XMAX = XMAX*REC
*       END IF
*   END IF
*   X( J ) = ZLADIV( X( J ), TJJS )
*   ELSE IF( TJJ.GT.ZERO ) THEN
*
*       0 < abs(A(j,j)) <= SMLNUM:
*
*       IF( XJ.GT.TJJ*BIGNUM ) THEN
*
*           Scale x by (1/abs(x(j)))*abs(A(j,j))*BIGNUM.
*
*           REC = ( TJJ*BIGNUM ) / XJ
*           CALL ZDSCAL( N, REC, X, 1 )
*           SCALE = SCALE*REC
*           XMAX = XMAX*REC
*       END IF
*       X( J ) = ZLADIV( X( J ), TJJS )
*   ELSE
*
*       A(j,j) = 0: Set x(1:n) = 0, x(j) = 1, and
*       scale = 0 and compute a solution to A**T *x = 0.
*
*       DO 150 I = 1, N
*           X( I ) = ZERO
*150      CONTINUE
*       X( J ) = ONE
*       SCALE = ZERO
*       XMAX = ZERO
*   END IF
*160      CONTINUE
*   ELSE
*
*       Compute x(j) := x(j) / A(j,j) - CSUMJ if the dot
*       product has already been divided by 1/A(j,j).
*
*       X( J ) = ZLADIV( X( J ), TJJS ) - CSUMJ
*   END IF
*       XMAX = MAX( XMAX, CABS1( X( J ) ) )
*170      CONTINUE
*
*   ELSE
*
*       Solve A**H * x = b
*

```

```

DO 220 J = JFIRST, JLAST, JINC
*
*      Compute x(j) = b(j) - sum A(k,j)*x(k).
*                      k<>j
*
      XJ = CABS1( X( J ) )
      USCAL = TSCAL
      REC = ONE / MAX( XMAX, ONE )
      IF( CNORM( J ).GT.( BIGNUM-XJ )*REC ) THEN
*
*      If x(j) could overflow, scale x by 1/(2*XMAX).
*
      REC = REC*HALF
      IF( NOUNIT ) THEN
          TJJS = DCONJG( A( J, J ) )*TSCAL
      ELSE
          TJJS = TSCAL
      END IF
      TJJ = CABS1( TJJS )
      IF( TJJ.GT.ONE ) THEN
*
*      Divide by A(j,j) when scaling x if A(j,j) > 1.
*
      REC = MIN( ONE, REC*TJJ )
      USCAL = ZLADIV( USCAL, TJJS )
      END IF
      IF( REC.LT.ONE ) THEN
          CALL ZDSCAL( N, REC, X, 1 )
          SCALE = SCALE*REC
          XMAX = XMAX*REC
      END IF
      END IF
*
      CSUMJ = ZERO
      IF( USCAL.EQ.DCMPLX( ONE ) ) THEN
*
*      If the scaling needed for A in the dot product is 1,
*      call ZDOTC to perform the dot product.
*
      IF( UPPER ) THEN
          CSUMJ = ZDOTC( J-1, A( 1, J ), 1, X, 1 )
      ELSE IF( J.LT.N ) THEN
          CSUMJ = ZDOTC( N-J, A( J+1, J ), 1, X( J+1 ), 1 )
      END IF
      ELSE
*
*      Otherwise, use in-line code for the dot product.
*
      IF( UPPER ) THEN
          DO 180 I = 1, J - 1

```

```

      CSUMJ = CSUMJ + ( DCONJG( A( I, J ) ) * USCAL ) *
$      X( I )
180      CONTINUE
      ELSE IF( J.LT.N ) THEN
      DO 190 I = J + 1, N
      CSUMJ = CSUMJ + ( DCONJG( A( I, J ) ) * USCAL ) *
$      X( I )
190      CONTINUE
      END IF
END IF

*
IF( USCAL.EQ.DCMPLX( TSCAL ) ) THEN
*
*      Compute x(j) := ( x(j) - CSUMJ ) / A(j,j) if 1/A(j,j)
*      was not used to scale the dotproduct.
*
      X( J ) = X( J ) - CSUMJ
      XJ = CABS1( X( J ) )
      IF( NOUNIT ) THEN
      TJJS = DCONJG( A( J, J ) ) * TSCAL
      ELSE
      TJJS = TSCAL
      IF( TSCAL.EQ.ONE )
      GO TO 210
$      END IF
*
*      Compute x(j) = x(j) / A(j,j), scaling if necessary.
*
*      TJJ = CABS1( TJJS )
*      IF( TJJ.GT.SMLNUM ) THEN
*
*          abs(A(j,j)) > SMLNUM:
*
*      IF( TJJ.LT.ONE ) THEN
*          IF( XJ.GT.TJJ*BIGNUM ) THEN
*
*              Scale X by 1/abs(x(j)).
*
*              REC = ONE / XJ
*              CALL ZDSCAL( N, REC, X, 1 )
*              SCALE = SCALE*REC
*              XMAX = XMAX*REC
*          END IF
*      END IF
*      X( J ) = ZLADIV( X( J ), TJJS )
*      ELSE IF( TJJ.GT.ZERO ) THEN
*
*          0 < abs(A(j,j)) <= SMLNUM:
*
*      IF( XJ.GT.TJJ*BIGNUM ) THEN

```

```

*
*           Scale x by (1/abs(x(j)))*abs(A(j,j))*BIGNUM.
*
*           REC = ( TJJ*BIGNUM ) / XJ
*           CALL ZDSCAL( N, REC, X, 1 )
*           SCALE = SCALE*REC
*           XMAX = XMAX*REC
*           END IF
*           X( J ) = ZLADIV( X( J ), TJJS )
*       ELSE
*
*           A(j,j) = 0:  Set x(1:n) = 0, x(j) = 1, and
*           scale = 0 and compute a solution to A**H *x = 0.
*
*           DO 200 I = 1, N
*               X( I ) = ZERO
*       200      CONTINUE
*           X( J ) = ONE
*           SCALE = ZERO
*           XMAX = ZERO
*           END IF
*       210      CONTINUE
*       ELSE
*
*           Compute x(j) := x(j) / A(j,j) - CSUMJ if the dot
*           product has already been divided by 1/A(j,j).
*
*           X( J ) = ZLADIV( X( J ), TJJS ) - CSUMJ
*           END IF
*           XMAX = MAX( XMAX, CABS1( X( J ) ) )
*       220      CONTINUE
*           END IF
*           SCALE = SCALE / TSCAL
*       END IF
*
*       Scale the column norms by 1/TSCAL for return.
*
*       IF( TSCAL.NE.ONE ) THEN
*           CALL DSCAL( N, ONE / TSCAL, CNORM, 1 )
*       END IF
*
*       RETURN
*
*       End of ZLATRS
*
*       END

```

— LAPACK zlatrs —

```

(let* ((zero 0.0d0) (half 0.5d0) (one 1.0d0) (two 2.0d0))
  (declare (type (double-float 0.0d0 0.0d0) zero)
    (type (double-float 0.5d0 0.5d0) half) (type (double-float 1.0d0 1.0d0) one)
    (type (double-float 2.0d0 2.0d0) two) (ignorable zero half one two))
  (defun zlatrs (uplo trans diag normin n a lda x scale cnorm info)
    (declare (type (simple-array character (*)) normin diag trans uplo)
      (type (f2cl-lib:integer4) info lda n)
      (type (array f2cl-lib:complex16 (*)) x a) (type (double-float) scale)
      (type (array double-float (*)) cnorm))
    (f2cl-lib:with-multi-array-data
      ((cnorm double-float cnorm-%data%
        cnorm-%offset%)
        (a f2cl-lib:complex16 a-%data% a-%offset%)
        (x f2cl-lib:complex16 x-%data% x-%offset%)
        (uplo character uplo-%data% uplo-%offset%)
        (trans character trans-%data% trans-%offset%)
        (diag character diag-%data% diag-%offset%)
        (normin character normin-%data% normin-%offset%))
      (labels
        ((cabs1 (zdum) (+ (abs (f2cl-lib:db1e zdum))
          (abs (f2cl-lib:dimag zdum)))))
          (cabs2 (zdum)
            (+ (abs (/ (f2cl-lib:db1e zdum) 2.0d0))
              (abs (/ (f2cl-lib:dimag zdum) 2.0d0)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
            (values double-float &rest t)) cabs1))
          (declare
            (ftype (function (f2cl-lib:complex16)
              (values double-float &rest t)) cabs2))
          (prog
            ((csumj #C(0.0d0 0.0d0)) (tjjs #C(0.0d0 0.0d0))
              (uscal #C(0.0d0 0.0d0))
              (zdum #C(0.0d0 0.0d0)) (bignum 0.0d0) (grow 0.0d0) (rec 0.0d0)
              (smlnum 0.0d0) (tjj 0.0d0) (tmax 0.0d0) (tscal 0.0d0) (xbnd 0.0d0)
              (xj 0.0d0) (xmax 0.0d0) (i 0) (imax 0) (j 0) (jfirst 0)
              (jinc 0) (jlast 0)
              (notran nil) (nounit nil) (upper nil))
            (declare (type (f2cl-lib:complex16) zdum uscal tjjs csumj)
              (type (double-float) xmax xj xbnd tscal tmax tjj
                smlnum rec grow bignum)
              (type (f2cl-lib:integer4) jlast jinc jfirst j imax i)
              (type f2cl-lib:logical upper nounit notran))
            (setf info 0)
            (setf upper
              (multiple-value-bind (ret-val var-0 var-1) (lsame uplo "U")
                (declare (ignore var-1)) (when var-0 (setf uplo var-0)) ret-val)))

```

```

(setf notran
  (multiple-value-bind (ret-val var-0 var-1) (lsame trans "N")
    (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val))
(setf nunit
  (multiple-value-bind (ret-val var-0 var-1) (lsame diag "N")
    (declare (ignore var-1)) (when var-0 (setf diag var-0)) ret-val))
(cond
  ((and (not upper)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame uplo "L")
        (declare (ignore var-1))
        (when var-0 (setf uplo var-0)) ret-val)))
    (setf info -1))
  ((and (not notran)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame trans "T")
        (declare (ignore var-1))
        (when var-0 (setf trans var-0)) ret-val))
      (not
        (multiple-value-bind (ret-val var-0 var-1) (lsame trans "C")
          (declare (ignore var-1))
          (when var-0 (setf trans var-0)) ret-val))))
    (setf info -2))
  ((and (not nunit)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame diag "U")
        (declare (ignore var-1))
        (when var-0 (setf diag var-0)) ret-val)))
    (setf info -3))
  ((and
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame normin "Y")
        (declare (ignore var-1))
        (when var-0 (setf normin var-0)) ret-val))
      (not
        (multiple-value-bind (ret-val var-0 var-1) (lsame normin "N")
          (declare (ignore var-1))
          (when var-0 (setf normin var-0)) ret-val))))
    (setf info -4))
  ((< n 0) (setf info -5))
  ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
    (setf info -7)))
(cond ((/= info 0)
  (xerbla "ZLATRS" (f2cl-lib:int-sub info)) (go end_label)))
(if (= n 0) (go end_label)) (setf smlnum (dlamch "Safe minimum"))
(setf bignum (/ one smlnum))
(multiple-value-bind (var-0 var-1)
  (dlabad smlnum bignum) (declare (ignore)
    (when var-0 (setf smlnum var-0)) (when var-1 (setf bignum var-1))))
(setf smlnum (/ smlnum (dlamch "Precision")))

```

```

(setf bignum (/ one smlnum))
(setf scale one)
(cond
  ((multiple-value-bind (ret-val var-0 var-1) (lsame normin "N")
    (declare (ignore var-1)) (when var-0 (setf normin var-0)) ret-val)
    (cond
      (upper
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
          ((> j n) nil)
          (tagbody
            (setf (f2cl-lib:fref cnorm-%data% (j)
              ((1 *)) cnorm-%offset%)
              (dzasum (f2cl-lib:int-sub j 1)
                (f2cl-lib:array-slice a-%data%
                  f2cl-lib:complex16 (1 j)
                  ((1 lda) (1 *)) a-%offset%)
                1))
              label10))))
        (t
          (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
            ((> j
              (f2cl-lib:int-add n (f2cl-lib:int-sub 1)))
              nil)
            (tagbody
              (setf (f2cl-lib:fref cnorm-%data% (j)
                ((1 *)) cnorm-%offset%)
                (dzasum (f2cl-lib:int-sub n j)
                  (f2cl-lib:array-slice a-%data%
                    f2cl-lib:complex16 ((+ j 1) j)
                    ((1 lda) (1 *)) a-%offset%)
                  1))
                label20))
              (setf (f2cl-lib:fref cnorm-%data% (n)
                ((1 *)) cnorm-%offset%) zero))))))
      (setf imax
        (multiple-value-bind (ret-val var-0 var-1 var-2) (idamax n cnorm 1)
          (declare (ignore var-1 var-2)) (when var-0 (setf n var-0)) ret-val))
      (setf tmax (f2cl-lib:fref cnorm-%data% (imax) ((1 *)) cnorm-%offset%))
      (cond ((<= tmax (* bignum half)) (setf tscal one))
        (t (setf tscal (/ half (* smlnum tmax))))
        (multiple-value-bind (var-0 var-1 var-2 var-3)
          (dscal n tscal cnorm 1)
          (declare (ignore var-2 var-3)) (when var-0 (setf n var-0))
          (when var-1 (setf tscal var-1))))))
      (setf xmax zero)
      (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
        ((> j n) nil)
        (tagbody
          (setf xmax
            (max xmax (cabs2 (f2cl-lib:fref x-%data% (j)

```

```

((1 *)) x-%offset%))))
label30))
(setf xbnd xmax)
(cond
  (notran
    (tagbody
      (cond (upper (setf jfirst n) (setf jlast 1) (setf jinc -1))
            (t (setf jfirst 1) (setf jlast n) (setf jinc 1)))
      (cond ((/= tscal one) (setf grow zero) (go label60)))
      (cond
        (nunit (setf grow (/ half (max xbnd smlnum))) (setf xbnd grow)
          (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
            (> j jlast) nil)
          (tagbody
            (if (<= grow smlnum) (go label60))
            (setf tjjs (f2cl-lib:fref a-%data% (j j)
              ((1 lda) (1 *)) a-%offset%))
            (setf tjj (cabs1 tjjs))
            (cond ((>= tjj smlnum)
              (setf xbnd (min xbnd (* (min one tjj) grow))))
              (t (setf xbnd zero)))
            (cond
              (>= (+ tjj (f2cl-lib:fref cnorm-%data% (j)
                ((1 *)) cnorm-%offset%)))) smlnum)
              (setf grow
                (* grow
                  (/ tjj
                    (+ tjj
                      (f2cl-lib:fref cnorm-%data% (j)
                        ((1 *)) cnorm-%offset%))))))
              (t (setf grow zero)))
            label40))
          (setf grow xbnd))
        (t (setf grow (min one (/ half (max xbnd smlnum))))
          (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
            (> j jlast) nil)
          (tagbody
            (if (<= grow smlnum) (go label60))
            (setf grow
              (* grow
                (/ one
                  (+ one (f2cl-lib:fref cnorm-%data% (j)
                    ((1 *)) cnorm-%offset%))))))
            label50))))
          label60))
      (t
        (tagbody
          (cond (upper (setf jfirst 1) (setf jlast n) (setf jinc 1))
                (t (setf jfirst n) (setf jlast 1) (setf jinc -1)))
          (cond ((/= tscal one) (setf grow zero) (go label90)))

```

```

(cond
  (nounit (setf grow (/ half (max xbnd smlnum))) (setf xbnd grow)
    (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
      (> j jlast) nil)
    (tagbody
      (if (<= grow smlnum) (go label90))
      (setf xj
        (+ one (f2cl-lib:fref cnorm-%data% (j)
          ((1 *)) cnorm-%offset%)))
      (setf grow (min grow (/ xbnd xj)))
      (setf tjjs (f2cl-lib:fref a-%data% (j j)
        ((1 lda) (1 *)) a-%offset%))
      (setf tjj (cabs1 tjjs))
      (cond (>= tjj smlnum)
        (if (> xj tjj)
          (setf xbnd (* xbnd (/ tjj xj)))))
      (t (setf xbnd zero)))
      label70))
  (setf grow (min grow xbnd)))
(t (setf grow (min one (/ half (max xbnd smlnum))))
  (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
    (> j jlast) nil)
    (tagbody
      (if (<= grow smlnum) (go label90))
      (setf xj
        (+ one (f2cl-lib:fref cnorm-%data% (j)
          ((1 *)) cnorm-%offset%)))
      (setf grow (/ grow xj)) label80))))
label90)))
(cond
  (> (* grow tscal) smlnum)
  (multiple-value-bind (var-0 var-1 var-2 var-3 var-4
    var-5 var-6 var-7)
    (ztrsv uplo trans diag n a lda x 1)
    (declare (ignore var-4 var-6 var-7))
    (when var-0 (setf uplo var-0)) (when var-1 (setf trans var-1))
    (when var-2 (setf diag var-2)) (when var-3 (setf n var-3))
    (when var-5 (setf lda var-5)))
  (t
    (cond
      (> xmax (* bignum half)) (setf scale (/ (* bignum half) xmax))
      (multiple-value-bind (var-0 var-1 var-2 var-3)
        (zdscal n scale x 1)
        (declare (ignore var-2 var-3)) (when var-0 (setf n var-0))
        (when var-1 (setf scale var-1)))
      (setf xmax bignum))
      (t (setf xmax (* xmax two))))
    (cond
      (notran
        (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))

```

```

(<> j jlast) nil)
(tagbody
  (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%)))
  (cond
    (nunit
      (setf tjjs
        (* (f2cl-lib:fref a-%data% (j j)
          ((1 lda) (1 *)) a-%offset%) tscal)))
      (t (setf tjjs (coerce tscal 'f2cl-lib:complex16))
        (if (= tscal one) (go label110))))
    (setf tjj (cabs1 tjjs))
    (cond
      (<> tjj smlnum)
      (cond
        (<< tjj one)
        (cond
          (<> xj (* tjj bignum)) (setf rec (/ one xj))
          (multiple-value-bind (var-0 var-1 var-2
            var-3)
            (zdscal n rec x 1)
            (declare (ignore var-2 var-3))
            (when var-0 (setf n var-0))
            (when var-1 (setf rec var-1))
            (setf scale (* scale rec))
            (setf xmax (* xmax rec))))))
          (setf (f2cl-lib:fref x-%data% (j)
            ((1 *)) x-%offset%)
            (zladd (f2cl-lib:fref x-%data% (j)
              ((1 *)) x-%offset%) tjjs))
          (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
            ((1 *)) x-%offset%)))
          (<> tjj zero)
          (cond
            (<> xj (* tjj bignum))
            (setf rec (/ (* tjj bignum) xj))
            (cond
              (<> (f2cl-lib:fref cnorm (j) ((1 *))) one)
              (setf rec
                (/ rec
                  (f2cl-lib:fref cnorm-%data% (j)
                    ((1 *)) cnorm-%offset%))))
              (multiple-value-bind (var-0 var-1 var-2 var-3)
                (zdscal n rec x 1)
                (declare (ignore var-2 var-3))
                (when var-0 (setf n var-0))
                (when var-1 (setf rec var-1))
                (setf scale (* scale rec))
                (setf xmax (* xmax rec))))
              (setf (f2cl-lib:fref x-%data% (j)

```

```

        ((1 *)) x-%offset%)
      (zladv (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%) tjs))
    (setf xj (cabs1
      (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%))))
    (t
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf (f2cl-lib:fref x-%data% (i)
            ((1 *)) x-%offset%)
            (coerce zero 'f2cl-lib:complex16))
            label110))
          (setf (f2cl-lib:fref x-%data% (j)
            ((1 *)) x-%offset%)
            (coerce one 'f2cl-lib:complex16))
            (setf xj one)
            (setf scale zero)
            (setf xmax zero)))
          label110
        (cond
          ((> xj one) (setf rec (/ one xj))
            (cond
              ((> (f2cl-lib:fref cnorm (j) ((1 *))
                (* (+ bignum (- xmax)) rec))
                (setf rec (* rec half))
                (multiple-value-bind (var-0 var-1 var-2 var-3)
                  (zdscal n rec x 1)
                  (declare (ignore var-2 var-3))
                  (when var-0 (setf n var-0))
                  (when var-1 (setf rec var-1)))
                (setf scale (* scale rec))))
              ((> (* xj (f2cl-lib:fref cnorm (j)
                ((1 *))) (+ bignum (- xmax)))
                (multiple-value-bind (var-0 var-1 var-2 var-3)
                  (zdscal n half x 1)
                  (declare (ignore var-2 var-3))
                  (when var-0 (setf n var-0))
                  (when var-1 (setf half var-1)))
                (setf scale (* scale half))))
            (cond
              (upper
                (cond
                  ((> j 1)
                    (zaxpy (f2cl-lib:int-sub j 1)
                      (* (- (f2cl-lib:fref x-%data% (j)
                        ((1 *)) x-%offset%)) tscal)
                      (f2cl-lib:array-slice a-%data%
                        f2cl-lib:complex16 (1 j)

```

```

      ((1 lda) (1 *)) a-%offset%)
      1 x 1)
      (setf i (izamax (f2cl-lib:int-sub j 1) x 1))
      (setf xmax
        (cabs1 (f2cl-lib:fref x-%data% (i)
          ((1 *)) x-%offset%))))))
    (t
      (cond
        ((< j n)
          (zaxpy (f2cl-lib:int-sub n j)
            (* (- (f2cl-lib:fref x-%data% (j)
              ((1 *)) x-%offset%)) tscal)
            (f2cl-lib:array-slice a-%data%
              f2cl-lib:complex16 ((+ j 1) j)
              ((1 lda) (1 *)) a-%offset%)
            1
            (f2cl-lib:array-slice x-%data%
              f2cl-lib:complex16 ((+ j 1) ((1 *))
                x-%offset%)
              1)
            (setf i
              (f2cl-lib:int-add j
                (izamax (f2cl-lib:int-sub n j)
                  (f2cl-lib:array-slice x-%data%
                    f2cl-lib:complex16 ((+ j 1)
                      ((1 *)) x-%offset%)
                    1)))
              (setf xmax
                (cabs1 (f2cl-lib:fref x-%data% (i)
                  ((1 *)) x-%offset%))))))
          label120)))
    ((multiple-value-bind (ret-val var-0 var-1) (lsame trans "T")
      (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val)
      (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
        (> j jlast) nil)
      (tagbody
        (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
          ((1 *)) x-%offset%)))
        (setf uscal (coerce tscal 'f2cl-lib:complex16))
        (setf rec (/ one (max xmax one)))
        (cond
          ((> (f2cl-lib:fref cnorm (j)
            ((1 *)) (* (+ bignum (- xj)) rec))
            (setf rec (* rec half))
          (cond
            (nunit
              (setf tjjs
                (* (f2cl-lib:fref a-%data% (j j)
                  ((1 lda) (1 *)) a-%offset%)
                  tscal)))

```



```

(t (setf tjjs
  (coerce tscal 'f2cl-lib:complex16))))
(setf tjj (cabs1 tjjs))
(cond
  ((> tjj one) (setf rec (min one (* rec tjj)))
    (setf uscal (zladiv uscal tjjs))))
(cond
  ((< rec one)
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (zdscal n rec x 1)
      (declare (ignore var-2 var-3))
      (when var-0 (setf n var-0))
      (when var-1 (setf rec var-1)))
    (setf scale (* scale rec))
    (setf xmax (* xmax rec))))))
(setf csumj (coerce zero 'f2cl-lib:complex16))
(cond
  ((= uscal (f2cl-lib:dcmplx one))
    (cond
      (upper
        (setf csumj
          (zdotu (f2cl-lib:int-sub j 1)
            (f2cl-lib:array-slice a-%data%
              f2cl-lib:complex16 (1 j)
              ((1 lda) (1 *)) a-%offset%
              1 x 1)))
        ((< j n)
          (setf csumj
            (zdotu (f2cl-lib:int-sub n j)
              (f2cl-lib:array-slice a-%data%
                f2cl-lib:complex16 ((+ j 1) j)
                ((1 lda) (1 *)) a-%offset%
                1
                (f2cl-lib:array-slice x-%data%
                  f2cl-lib:complex16 ((+ j 1))
                  ((1 *)) x-%offset%
                  1))))))
      (t
        (cond
          (upper
            (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
              ((> i
                (f2cl-lib:int-add j
                  (f2cl-lib:int-sub 1)))
                nil)
            (tagbody
              (setf csumj
                (+ csumj
                  (* (f2cl-lib:fref a-%data% (i j)
                    ((1 lda) (1 *)) a-%offset%

```

```

        uscal (f2cl-lib:fref x-%data% (i)
              ((1 *)) x-%offset%)))
      label130)))
    ((< j n)
      (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                    (f2cl-lib:int-add i 1))
        ((> i
          n)
          nil)
      (tagbody
        (setf csumj
          (+ csumj
            (* (f2cl-lib:fref a-%data% (i j)
                          ((1 lda) (1 *)) a-%offset%)
              uscal (f2cl-lib:fref x-%data% (i)
                    ((1 *)) x-%offset%)))
            label140))))))
    (cond
      ((= uscal (f2cl-lib:dcmplx tscal))
        (tagbody
          (setf (f2cl-lib:fref x-%data% (j)
                    ((1 *)) x-%offset%)
            (- (f2cl-lib:fref x-%data% (j)
                          ((1 *)) x-%offset%) csumj))
          (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
                    ((1 *)) x-%offset%)))
          (cond
            (nounit
              (setf tjjs
                (* (f2cl-lib:fref a-%data% (j j)
                          ((1 lda) (1 *)) a-%offset%)
                  tscal)))
            (t (setf tjjs
              (coerce tscal 'f2cl-lib:complex16))
              (if (= tscal one) (go label160))))
          (setf tjj (cabs1 tjjs))
          (cond
            ((> tjj smlnum)
              (cond
                ((< tjj one)
                  (cond
                    ((> xj (* tjj bignum))
                      (setf rec (/ one xj))
                      (multiple-value-bind (var-0 var-1
                                            var-2 var-3)
                        (zdscal n rec x 1)
                        (declare (ignore var-2 var-3))
                        (when var-0 (setf n var-0))
                        (when var-1 (setf rec var-1)))
                      (setf scale (* scale rec))

```

```

        (setf xmax (* xmax rec))))))
      (setf (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%)
        (zladv (f2cl-lib:fref x-%data% (j)
          ((1 *)) x-%offset%) tjjs)))
    (> tjj zero)
    (cond
      ((> xj (* tjj bignum))
       (setf rec (/ (* tjj bignum) xj))
       (multiple-value-bind (var-0 var-1 var-2
         var-3)
         (zdscl n rec x 1)
         (declare (ignore var-2 var-3))
         (when var-0 (setf n var-0))
         (when var-1 (setf rec var-1)))
       (setf scale (* scale rec))
       (setf xmax (* xmax rec))))
      (setf (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%)
        (zladv (f2cl-lib:fref x-%data% (j)
          ((1 *)) x-%offset%) tjjs)))
    (t
     (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)
    (tagbody
      (setf (f2cl-lib:fref x-%data% (i)
        ((1 *)) x-%offset%)
        (coerce zero 'f2cl-lib:complex16))
      label150))
    (setf (f2cl-lib:fref x-%data% (j)
      ((1 *)) x-%offset%)
      (coerce one 'f2cl-lib:complex16))
    (setf scale zero) (setf xmax zero)))
  label160))
(t
 (setf (f2cl-lib:fref x-%data% (j)
  ((1 *)) x-%offset%)
  (- (zladv (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%) tjjs)
    csumj))))
(setf xmax
 (max xmax (cabs1
  (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%))))
label170)))
(t
 (f2cl-lib:fdo (j jfirst (f2cl-lib:int-add j jinc))
  (> j jlast) nil)
 (tagbody
  (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%))))

```

```

((1 *)) x-%offset%)))
(setf uscal (coerce tscal 'f2cl-lib:complex16))
(setf rec (/ one (max xmax one)))
(cond
  (> (f2cl-lib:fref cnorm (j) ((1 *)))
    (* (+ bignum (- xj)) rec))
  (setf rec (* rec half))
  (cond
    (nunit
      (setf tjjs
        (coerce
          (*
            (f2cl-lib:dconjg
              (f2cl-lib:fref a-%data% (j j)
                ((1 lda) (1 *)) a-%offset%)
              tscal)
            'f2cl-lib:complex16)))
      (t
        (setf tjjs
          (coerce tscal 'f2cl-lib:complex16))))
    (setf tjj (cabs1 tjjs))
    (cond
      (> tjj one) (setf rec (min one (* rec tjj)))
      (setf uscal (zladdv uscal tjjs))))
  (cond
    (< rec one)
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (zdscal n rec x 1)
      (declare (ignore var-2 var-3))
      (when var-0 (setf n var-0))
      (when var-1 (setf rec var-1))
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))))
(setf csumj (coerce zero 'f2cl-lib:complex16))
(cond
  (= uscal (f2cl-lib:dcmplx one))
  (cond
    (upper
      (setf csumj
        (zdadc (f2cl-lib:int-sub j 1)
          (f2cl-lib:array-slice a-%data%
            f2cl-lib:complex16 (1 j)
              ((1 lda) (1 *)) a-%offset%)
            1 x 1)))
    (< j n)
    (setf csumj
      (zdadc (f2cl-lib:int-sub n j)
        (f2cl-lib:array-slice a-%data%
          f2cl-lib:complex16 ((+ j 1) j)
            ((1 lda) (1 *)) a-%offset%)
          1 x 1)))
  )
)

```

```

1
(f2cl-lib:array-slice x-%data%
 f2cl-lib:complex16 ((+ j 1))
 ((1 *)) x-%offset%)
1))))))
(t
 (cond
  (upper
   (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i
      (f2cl-lib:int-add j
       (f2cl-lib:int-sub 1)))
      nil)
    (tagbody
     (setf csumj
      (+ csumj
       (*
        (f2cl-lib:dconjg
         (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%))
        uscal (f2cl-lib:fref x-%data% (i)
          ((1 *)) x-%offset%))))
      label180))))
    ((< j n)
     (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
      (f2cl-lib:int-add i 1))
      ((> i
        n)
        nil)
      (tagbody
       (setf csumj
        (+ csumj
         (*
          (f2cl-lib:dconjg
           (f2cl-lib:fref a-%data% (i j)
            ((1 lda) (1 *)) a-%offset%))
          uscal (f2cl-lib:fref x-%data% (i)
            ((1 *)) x-%offset%))))
        label190))))))
    (cond
     ((= uscal (f2cl-lib:dcmplx tscal))
      (tagbody
       (setf (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%)
        (- (f2cl-lib:fref x-%data% (j)
          ((1 *)) x-%offset%) csumj))
       (setf xj (cabs1 (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%))))
      (cond
       (nounit

```

```

(setf tjjs
  (coerce
    (*
      (f2cl-lib:dconjg
        (f2cl-lib:fref a-%data% (j j)
          ((1 lda) (1 *)) a-%offset%))
      tscal)
    'f2cl-lib:complex16)))
(t (setf tjjs
  (coerce tscal 'f2cl-lib:complex16))
  (if (= tscal one) (go label210))))
(setf tjj (cabs1 tjjs))
(cond
  (> tjj smlnum)
  (cond
    (< tjj one)
    (cond
      (> xj (* tjj bignum))
      (setf rec (/ one xj))
      (multiple-value-bind (var-0 var-1
        var-2 var-3)
        (zdscal n rec x 1)
        (declare (ignore var-2 var-3))
        (when var-0 (setf n var-0))
        (when var-1 (setf rec var-1))
        (setf scale (* scale rec))
        (setf xmax (* xmax rec))))))
    (setf (f2cl-lib:fref x-%data% (j)
      ((1 *)) x-%offset%)
      (zladdv (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%) tjjs)))
  (> tjj zero)
  (cond
    (> xj (* tjj bignum))
    (setf rec (/ (* tjj bignum) xj))
    (multiple-value-bind (var-0 var-1 var-2
      var-3)
      (zdscal n rec x 1)
      (declare (ignore var-2 var-3))
      (when var-0 (setf n var-0))
      (when var-1 (setf rec var-1))
      (setf scale (* scale rec))
      (setf xmax (* xmax rec))))
    (setf (f2cl-lib:fref x-%data% (j)
      ((1 *)) x-%offset%)
      (zladdv (f2cl-lib:fref x-%data% (j)
        ((1 *)) x-%offset%) tjjs)))
  (t
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)

```

```

(tagbody
  (setf (f2cl-lib:fref x-%data% (i)
    ((1 *)) x-%offset%)
    (coerce zero 'f2cl-lib:complex16))
    label200))
  (setf (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%)
    (coerce one 'f2cl-lib:complex16))
    (setf scale zero) (setf xmax zero)))
  label210))
(t
  (setf (f2cl-lib:fref x-%data% (j)
    ((1 *)) x-%offset%)
    (- (zladd (f2cl-lib:fref x-%data% (j)
      ((1 *)) x-%offset%) tjj)
      csumj))))
  (setf xmax
    (max xmax (cabs1 (f2cl-lib:fref x-%data% (j)
      ((1 *)) x-%offset%))))
    label220))))
  (setf scale (/ scale tscal))))
(cond
  ((/= tscal one)
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (dscal n (/ one tscal) cnorm 1)
      (declare (ignore var-1 var-2 var-3))
      (when var-0 (setf n var-0))))))
  (go end_label) end_label
  (return
    (values uplo trans diag normin n nil lda nil scale nil info))))
)))

```

zrot LAPACK

— zrot.input —

```

)set break resume
)sys rm -f zrot.output
)spool zrot.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zrot.help —

```
=====
zrot examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====
```

```
SUBROUTINE ZROT( N, CX, INCX, CY, INCY, C, S )
```

```
.. Scalar Arguments ..
```

```
INTEGER          INCX, INCY, N
```

```
DOUBLE PRECISION C
```

```
COMPLEX*16       S
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       CX( * ), CY( * )
```

```
..
```

Purpose:

```
=====
```

ZROT applies a plane rotation, where the cos (C) is real and the sin (S) is complex, and the vectors CX and CY are complex.

Arguments:

```
=====
```

[in] N

N is INTEGER

The number of elements in the vectors CX and CY.

[in,out] CX

CX is COMPLEX*16 array, dimension (N)

On input, the vector X.

On output, CX is overwritten with $C*X + S*Y$.

[in] INCX

INCX is INTEGER
The increment between successive values of CY. INCX <> 0.

[in,out] CY

CY is COMPLEX*16 array, dimension (N)
On input, the vector Y.
On output, CY is overwritten with $-\text{CONJG}(S)*X + C*Y$.

[in] INCY

INCY is INTEGER
The increment between successive values of CY. INCX <> 0.

[in] C

C is DOUBLE PRECISION

[in] S

S is COMPLEX*16
C and S define a rotation

$$\begin{bmatrix} C & S \\ -\text{conjg}(S) & C \end{bmatrix}$$
 where $C*C + S*\text{CONJG}(S) = 1.0$.

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zrot.f —

```
* =====
* SUBROUTINE ZROT( N, CX, INCX, CY, INCY, C, S )
*
* -- LAPACK auxiliary routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
```

```

* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*   November 2011
*
*   .. Scalar Arguments ..
*     INTEGER          INCX, INCY, N
*     DOUBLE PRECISION C
*     COMPLEX*16       S
*
*   ..
*   .. Array Arguments ..
*     COMPLEX*16       CX( * ), CY( * )
*
*   ..
*
* =====
*
*   .. Local Scalars ..
*     INTEGER          I, IX, IY
*     COMPLEX*16       STEMP
*
*   ..
*   .. Intrinsic Functions ..
*     INTRINSIC        DCONJG
*
*   ..
*   .. Executable Statements ..
*
*     IF( N.LE.0 )
*       $ RETURN
*     IF( INCX.EQ.1 .AND. INCY.EQ.1 )
*       $ GO TO 20
*
*   ..
*   .. Code for unequal increments or equal increments not equal to 1
*
*     IX = 1
*     IY = 1
*     IF( INCX.LT.0 )
*       $ IX = ( -N+1 )*INCX + 1
*     IF( INCY.LT.0 )
*       $ IY = ( -N+1 )*INCY + 1
*     DO 10 I = 1, N
*       STEMP = C*CX( IX ) + S*CY( IY )
*       CY( IY ) = C*CY( IY ) - DCONJG( S )*CX( IX )
*       CX( IX ) = STEMP
*       IX = IX + INCX
*       IY = IY + INCY
* 10 CONTINUE
*     RETURN
*
*   ..
*   .. Code for both increments equal to 1
*
* 20 CONTINUE
*     DO 30 I = 1, N
*       STEMP = C*CX( I ) + S*CY( I )

```

```

      CY( I ) = C*CY( I ) - DCONJG( S )*CX( I )
      CX( I ) = STEMP
30 CONTINUE
      RETURN
      END

```

— LAPACK zrot —

```

(defun zrot (n cx incx cy incy c s)
  (declare (type (f2cl-lib:integer4) incy incx n)
    (type (array f2cl-lib:complex16 (*)) cy cx) (type (double-float) c)
    (type (f2cl-lib:complex16) s))
  (f2cl-lib:with-multi-array-data
    ((cx f2cl-lib:complex16 cx-%data% cx-%offset%)
     (cy f2cl-lib:complex16 cy-%data% cy-%offset%))
    (prog
      ((stemp #C(0.0d0 0.0d0)) (i 0) (ix 0) (iy 0))
      (declare (type (f2cl-lib:complex16) stemp)
        (type (f2cl-lib:integer4) iy ix i))
      (if (<= n 0) (go end_label))
      (if (and (= incx 1) (= incy 1)) (go label20))
      (setf ix 1) (setf iy 1)
      (if (< incx 0)
        (setf ix
          (f2cl-lib:int-add (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incx) 1)))
      (if (< incy 0)
        (setf iy
          (f2cl-lib:int-add (f2cl-lib:int-mul (f2cl-lib:int-sub 1 n) incy) 1)))
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)
        (tagbody
          (setf stemp
            (+ (* c (f2cl-lib:fref cx-%data% (ix) ((1 *)) cx-%offset%))
              (* s (f2cl-lib:fref cy-%data% (iy) ((1 *)) cy-%offset%))))
          (setf (f2cl-lib:fref cy-%data% (iy) ((1 *)) cy-%offset%)
            (- (* c (f2cl-lib:fref cy-%data% (iy) ((1 *)) cy-%offset%)
              (* (f2cl-lib:dconjg s)
                (f2cl-lib:fref cx-%data% (ix) ((1 *)) cx-%offset%))))
          (setf (f2cl-lib:fref cx-%data% (ix) ((1 *))
            cx-%offset%) stemp)
          (setf ix (f2cl-lib:int-add ix incx))
          (setf iy (f2cl-lib:int-add iy incy))
          label10))
      (go end_label) label20
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        ((> i n) nil)

```

```

(tagbody
  (setf stemp
    (+ (* c (f2cl-lib:fref cx-%data% (i) ((1 *)) cx-%offset%))
      (* s (f2cl-lib:fref cy-%data% (i) ((1 *)) cy-%offset%))))
  (setf (f2cl-lib:fref cy-%data% (i) ((1 *)) cy-%offset%)
    (- (* c (f2cl-lib:fref cy-%data% (i) ((1 *)) cy-%offset%)
      (* (f2cl-lib:dconjg s)
        (f2cl-lib:fref cx-%data% (i) ((1 *)) cx-%offset%))))
  (setf (f2cl-lib:fref cx-%data% (i) ((1 *)) cx-%offset%)
    stemp) label30)
)
(go end_label) end_label (return (values nil nil nil nil nil nil nil))
))

```

ztrevc LAPACK

— ztrevc.input —

```

)set break resume
)sys rm -f ztrevc.output
)spool ztrevc.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— ztrevc.help —

```

=====
ztrevc examples
=====

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```
SUBROUTINE ZTREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
                  LDVR, MM, M, WORK, RWORK, INFO )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          HOWMNY, SIDE
```

```
INTEGER           INFO, LDT, LDVL, LDVR, M, MM, N
```

```
..
```

```
.. Array Arguments ..
```

```
LOGICAL           SELECT( * )
```

```
DOUBLE PRECISION  RWORK( * )
```

```
COMPLEX*16        T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
```

```
$                WORK( * )
```

```
..
```

Purpose:

=====

ZTREVC computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T.

Matrices of this type are produced by the Schur factorization of a complex general matrix: $A = Q \cdot T \cdot Q^{*H}$, as computed by ZHSEQR.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

$$T \cdot x = w \cdot x, \quad (y^{*H}) \cdot T = w \cdot (y^{*H})$$

where y^{*H} denotes the conjugate transpose of the vector y.

The eigenvalues are not input to this routine, but are read directly from the diagonal of T.

This routine returns the matrices X and/or Y of right and left eigenvectors of T, or the products $Q \cdot X$ and/or $Q \cdot Y$, where Q is an input matrix. If Q is the unitary factor that reduces a matrix A to Schur form T, then $Q \cdot X$ and $Q \cdot Y$ are the matrices of right and left eigenvectors of A.

Arguments:

=====

[in] SIDE

SIDE is CHARACTER*1

= 'R': compute right eigenvectors only;

= 'L': compute left eigenvectors only;

= 'B': compute both right and left eigenvectors.

[in] HOWMNY

HOWMNY is CHARACTER*1
 = 'A': compute all right and/or left eigenvectors;
 = 'B': compute all right and/or left eigenvectors,
 backtransformed using the matrices supplied in
 VR and/or VL;
 = 'S': compute selected right and/or left eigenvectors,
 as indicated by the logical array SELECT.

[in] SELECT

SELECT is LOGICAL array, dimension (N)
 If HOWMNY = 'S', SELECT specifies the eigenvectors to be
 computed.
 The eigenvector corresponding to the j-th eigenvalue is
 computed if SELECT(j) = .TRUE..
 Not referenced if HOWMNY = 'A' or 'B'.

[in] N

N is INTEGER
 The order of the matrix T. $N \geq 0$.

[in,out] T

T is COMPLEX*16 array, dimension (LDT,N)
 The upper triangular matrix T. T is modified, but restored
 on exit.

[in] LDT

LDT is INTEGER
 The leading dimension of the array T. $LDT \geq \max(1, N)$.

[in,out] VL

VL is COMPLEX*16 array, dimension (LDVL,MM)
 On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must
 contain an N-by-N matrix Q (usually the unitary matrix Q of
 Schur vectors returned by ZHSEQR).
 On exit, if SIDE = 'L' or 'B', VL contains:
 if HOWMNY = 'A', the matrix Y of left eigenvectors of T;
 if HOWMNY = 'B', the matrix Q^*Y ;
 if HOWMNY = 'S', the left eigenvectors of T specified by
 SELECT, stored consecutively in the columns
 of VL, in the same order as their
 eigenvalues.
 Not referenced if SIDE = 'R'.

[in] LDVL

LDVL is INTEGER

The leading dimension of the array VL. LDVL \geq 1, and if
SIDE = 'L' or 'B', LDVL \geq N.

[in,out] VR

VR is COMPLEX*16 array, dimension (LDVR,MM)

On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must
contain an N-by-N matrix Q (usually the unitary matrix Q of
Schur vectors returned by ZHSEQR).

On exit, if SIDE = 'R' or 'B', VR contains:

if HOWMNY = 'A', the matrix X of right eigenvectors of T;

if HOWMNY = 'B', the matrix Q*X;

if HOWMNY = 'S', the right eigenvectors of T specified by
SELECT, stored consecutively in the columns
of VR, in the same order as their
eigenvalues.

Not referenced if SIDE = 'L'.

[in] LDVR

LDVR is INTEGER

The leading dimension of the array VR. LDVR \geq 1, and if
SIDE = 'R' or 'B'; LDVR \geq N.

[in] MM

MM is INTEGER

The number of columns in the arrays VL and/or VR. MM \geq M.

[out] M

M is INTEGER

The number of columns in the arrays VL and/or VR actually
used to store the eigenvectors. If HOWMNY = 'A' or 'B', M
is set to N. Each selected eigenvector occupies one
column.

[out] WORK

WORK is COMPLEX*16 array, dimension (2*N)

[out] RWORK

RWORK is DOUBLE PRECISION array, dimension (N)

[out] INFO

INFO is INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

Authors:
 =====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

Further Details:
 =====

The algorithm used in this program is basically backward (forward) substitution, with scaling to make the the code robust against possible overflow.

Each eigenvector is normalized so that the element of largest magnitude has magnitude 1; here the magnitude of a complex number (x,y) is taken to be $|x| + |y|$.

— ztrevc.f —

```
* =====
*      SUBROUTINE ZTREV( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
*      $                  LDVR, MM, M, WORK, RWORK, INFO )
*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
* .. Scalar Arguments ..
*      CHARACTER          HOWMNY, SIDE
*      INTEGER            INFO, LDT, LDVL, LDVR, M, MM, N
*
* ..
*
* .. Array Arguments ..
*      LOGICAL            SELECT( * )
*      DOUBLE PRECISION  RWORK( * )
*      COMPLEX*16         T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
*      $                  WORK( * )
*
* ..
```



```

*
* =====
*
* .. Parameters ..
DOUBLE PRECISION    ZERO, ONE
PARAMETER           ( ZERO = 0.0D+0, ONE = 1.0D+0 )
COMPLEX*16          CMZERO, CMONE
PARAMETER           ( CMZERO = ( 0.0D+0, 0.0D+0 ),
$                   CMONE = ( 1.0D+0, 0.0D+0 ) )
*
* ..
* .. Local Scalars ..
LOGICAL             ALLV, BOTHV, LEFTV, OVER, RIGHTV, SOMEV
INTEGER             I, II, IS, J, K, KI
DOUBLE PRECISION    OVFL, REMAX, SCALE, SMIN, SMLNUM, ULP, UNFL
COMPLEX*16          CDUM
*
* ..
* .. External Functions ..
LOGICAL             LSAME
INTEGER             IZAMAX
DOUBLE PRECISION    DLAMCH, DZASUM
EXTERNAL            LSAME, IZAMAX, DLAMCH, DZASUM
*
* ..
* .. External Subroutines ..
EXTERNAL            XERBLA, ZCOPY, ZDSCAL, ZGEMV, ZLATRS
*
* ..
* .. Intrinsic Functions ..
INTRINSIC           ABS, DBLE, DCMPLX, DCONJG, DIMAG, MAX
*
* ..
* .. Statement Functions ..
DOUBLE PRECISION    CABS1
*
* ..
* .. Statement Function definitions ..
CABS1( CDUM ) = ABS( DBLE( CDUM ) ) + ABS( DIMAG( CDUM ) )
*
* ..
* .. Executable Statements ..
*
* Decode and test the input parameters
*
*
*   BOTHV = LSAME( SIDE, 'B' )
*   RIGHTV = LSAME( SIDE, 'R' ) .OR. BOTHV
*   LEFTV = LSAME( SIDE, 'L' ) .OR. BOTHV
*
*
*   ALLV = LSAME( HOWMNY, 'A' )
*   OVER = LSAME( HOWMNY, 'B' )
*   SOMEV = LSAME( HOWMNY, 'S' )
*
*
* Set M to the number of columns required to store the selected
* eigenvectors.
*
*
*   IF( SOMEV ) THEN

```

```

        M = 0
        DO 10 J = 1, N
            IF( SELECT( J ) )
                $      M = M + 1
10      CONTINUE
        ELSE
            M = N
        END IF
*
        INFO = 0
        IF( .NOT.RIGHTV .AND. .NOT.LEFTV ) THEN
            INFO = -1
        ELSE IF( .NOT.ALLV .AND. .NOT.OVER .AND. .NOT.SOMEV ) THEN
            INFO = -2
        ELSE IF( N.LT.0 ) THEN
            INFO = -4
        ELSE IF( LDT.LT.MAX( 1, N ) ) THEN
            INFO = -6
        ELSE IF( LDVL.LT.1 .OR. ( LEFTV .AND. LDVL.LT.N ) ) THEN
            INFO = -8
        ELSE IF( LDVR.LT.1 .OR. ( RIGHTV .AND. LDVR.LT.N ) ) THEN
            INFO = -10
        ELSE IF( MM.LT.M ) THEN
            INFO = -11
        END IF
        IF( INFO.NE.0 ) THEN
            CALL XERBLA( 'ZTREVC', -INFO )
            RETURN
        END IF
*
*      Quick return if possible.
*
        IF( N.EQ.0 )
            $      RETURN
*
*      Set the constants to control overflow.
*
        UNFL = DLAMCH( 'Safe minimum' )
        OVFL = ONE / UNFL
        CALL DLABAD( UNFL, OVFL )
        ULP = DLAMCH( 'Precision' )
        SMLNUM = UNFL*( N / ULP )
*
*      Store the diagonal elements of T in working array WORK.
*
        DO 20 I = 1, N
            WORK( I+N ) = T( I, I )
20      CONTINUE
*
*      Compute 1-norm of each column of strictly upper triangular

```

```

*      part of T to control overflow in triangular solver.
*
      RWORK( 1 ) = ZERO
      DO 30 J = 2, N
        RWORK( J ) = DZASUM( J-1, T( 1, J ), 1 )
30    CONTINUE
*
      IF( RIGHTV ) THEN
*
*        Compute right eigenvectors.
*
        IS = M
        DO 80 KI = N, 1, -1
*
          IF( SOMEV ) THEN
            IF( .NOT.SELECT( KI ) )
$              GO TO 80
          END IF
          SMIN = MAX( ULP*( CABS1( T( KI, KI ) ) ), SMLNUM )
*
          WORK( 1 ) = CMONE
*
*        Form right-hand side.
*
          DO 40 K = 1, KI - 1
            WORK( K ) = -T( K, KI )
40        CONTINUE
*
*        Solve the triangular system:
*        (T(1:KI-1,1:KI-1) - T(KI,KI))*X = SCALE*WORK.
*
          DO 50 K = 1, KI - 1
            T( K, K ) = T( K, K ) - T( KI, KI )
            IF( CABS1( T( K, K ) ).LT.SMIN )
$              T( K, K ) = SMIN
50        CONTINUE
*
          IF( KI.GT.1 ) THEN
            CALL ZLATRS( 'Upper', 'No transpose', 'Non-unit', 'Y',
$              KI-1, T, LDT, WORK( 1 ), SCALE, RWORK,
$              INFO )
            WORK( KI ) = SCALE
          END IF
*
*        Copy the vector x or Q*x to VR and normalize.
*
          IF( .NOT.OVER ) THEN
            CALL ZCOPY( KI, WORK( 1 ), 1, VR( 1, IS ), 1 )
*
            II = IZAMAX( KI, VR( 1, IS ), 1 )

```

```

      REMAX = ONE / CABS1( VR( II, IS ) )
      CALL ZDSCAL( KI, REMAX, VR( 1, IS ), 1 )
*
      DO 60 K = KI + 1, N
        VR( K, IS ) = CMZERO
60    CONTINUE
      ELSE
        IF( KI.GT.1 )
          $      CALL ZGEMV( 'N', N, KI-1, CMONE, VR, LDVR, WORK( 1 ),
          $              1, DCMPLX( SCALE ), VR( 1, KI ), 1 )
*
          II = IZAMAX( N, VR( 1, KI ), 1 )
          REMAX = ONE / CABS1( VR( II, KI ) )
          CALL ZDSCAL( N, REMAX, VR( 1, KI ), 1 )
        END IF
*
*      Set back the original diagonal elements of T.
*
      DO 70 K = 1, KI - 1
        T( K, K ) = WORK( K+N )
70    CONTINUE
*
      IS = IS - 1
80    CONTINUE
      END IF
*
      IF( LEFTV ) THEN
*
*      Compute left eigenvectors.
*
      IS = 1
      DO 130 KI = 1, N
*
        IF( SOMEV ) THEN
          IF( .NOT.SELECT( KI ) )
            $      GO TO 130
          END IF
          SMIN = MAX( ULP*( CABS1( T( KI, KI ) ) ), SMLNUM )
*
          WORK( N ) = CMONE
*
*      Form right-hand side.
*
          DO 90 K = KI + 1, N
            WORK( K ) = -DCONJG( T( KI, K ) )
90        CONTINUE
*
*      Solve the triangular system:
*      (T(KI+1:N,KI+1:N) - T(KI,KI))**H * X = SCALE*WORK.
*

```

```

      DO 100 K = KI + 1, N
        T( K, K ) = T( K, K ) - T( KI, KI )
        IF( CABS1( T( K, K ) ).LT.SMIN )
          $      T( K, K ) = SMIN
100    CONTINUE
*
      IF( KI.LT.N ) THEN
        CALL ZLATRS( 'Upper', 'Conjugate transpose', 'Non-unit',
          $      'Y', N-KI, T( KI+1, KI+1 ), LDT,
          $      WORK( KI+1 ), SCALE, RWORK, INFO )
        WORK( KI ) = SCALE
      END IF
*
*      Copy the vector x or Q*x to VL and normalize.
*
      IF( .NOT.OVER ) THEN
        CALL ZCOPY( N-KI+1, WORK( KI ), 1, VL( KI, IS ), 1 )
*
        II = IZAMAX( N-KI+1, VL( KI, IS ), 1 ) + KI - 1
        REMAX = ONE / CABS1( VL( II, IS ) )
        CALL ZDSCAL( N-KI+1, REMAX, VL( KI, IS ), 1 )
*
        DO 110 K = 1, KI - 1
          VL( K, IS ) = CMZERO
110    CONTINUE
      ELSE
        IF( KI.LT.N )
          $      CALL ZGEMV( 'N', N, N-KI, CMONE, VL( 1, KI+1 ), LDVL,
          $      WORK( KI+1 ), 1, DCMPLX( SCALE ),
          $      VL( 1, KI ), 1 )
*
        II = IZAMAX( N, VL( 1, KI ), 1 )
        REMAX = ONE / CABS1( VL( II, KI ) )
        CALL ZDSCAL( N, REMAX, VL( 1, KI ), 1 )
      END IF
*
*      Set back the original diagonal elements of T.
*
      DO 120 K = KI + 1, N
        T( K, K ) = WORK( K+N )
120    CONTINUE
*
      IS = IS + 1
130    CONTINUE
      END IF
*
      RETURN
*
*      End of ZTREVCL
*

```

END

— LAPACK ztrevc —

```

(let*
  ((zero 0.0d0) (one 1.0d0)
   (cmzero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (cmone (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (double-float 0.0d0 0.0d0) zero)
            (type (double-float 1.0d0 1.0d0) one) (type (f2cl-lib:complex16) cmzero)
            (type (f2cl-lib:complex16) cmone) (ignorable zero one cmzero cmone))
  (defun ztrevc
    (side howmny select n t$ ldt vl ldvl vr ldvr mm m work rwork info)
    (declare (type (simple-array character (*)) howmny side)
              (type (array f2cl-lib:logical (*)) select)
              (type (f2cl-lib:integer4) info m mm ldvr ldvl ldt n)
              (type (array f2cl-lib:complex16 (*)) work vr vl t$)
              (type (array double-float (*)) rwork))
    (f2cl-lib:with-multi-array-data
      ((rwork double-float rwork-%data%
        rwork-%offset%)
       (t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
       (vl f2cl-lib:complex16 vl-%data% vl-%offset%)
       (vr f2cl-lib:complex16 vr-%data% vr-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%)
       (select f2cl-lib:logical select-%data% select-%offset%)
       (side character side-%data% side-%offset%)
       (howmny character howmny-%data% howmny-%offset%))
      (labels
        ((cabs1 (cdum)
          (+ (abs (f2cl-lib:dbble cdum)) (abs (f2cl-lib:dimag cdum)))))
        (declare
          (ftype (function (f2cl-lib:complex16)
                          (values double-float &rest t)) cabs1))
        (prog
          ((cdum #C(0.0d0 0.0d0)) (ovfl 0.0d0) (remax 0.0d0)
           (scale 0.0d0) (smin 0.0d0)
           (smlnum 0.0d0) (ulp 0.0d0) (unfl 0.0d0)
           (i 0) (ii 0) (is 0) (j 0) (k 0)
           (ki 0) (allv nil) (bothv nil) (leftv nil) (over nil) (rightv nil)
           (somev nil) (dcmplx$ 0.0))
          (declare (type (f2cl-lib:complex16) cdum)
                    (type (double-float) unfl ulp smlnum smin scale remax ovfl)
                    (type (f2cl-lib:integer4) ki k j is ii i)
                    (type f2cl-lib:logical somev rightv over leftv bothv allv)
                    (type (single-float) dcplx$))

```

```

(setf bothv
  (multiple-value-bind (ret-val var-0 var-1) (lsame side "B")
    (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
(setf rightv
  (or
    (multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
      (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
    bothv))
(setf leftv
  (or
    (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
      (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)
    bothv))
(setf allv
  (multiple-value-bind (ret-val var-0 var-1) (lsame howmny "A")
    (declare (ignore var-1)) (when var-0 (setf howmny var-0)) ret-val))
(setf over
  (multiple-value-bind (ret-val var-0 var-1) (lsame howmny "B")
    (declare (ignore var-1)) (when var-0 (setf howmny var-0)) ret-val))
(setf somev
  (multiple-value-bind (ret-val var-0 var-1) (lsame howmny "S")
    (declare (ignore var-1)) (when var-0 (setf howmny var-0)) ret-val))
(cond
  (somev (setf m 0)
    (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (if (f2cl-lib:fref select-%data% (j)
          ((1 *)) select-%offset%)
          (setf m (f2cl-lib:int-add m 1)))
        label10)))
  (t (setf m n)))
(setf info 0)
(cond ((and (not rightv) (not leftv)) (setf info -1))
  ((and (not allv) (not over) (not somev)) (setf info -2))
  ((< n 0) (setf info -4))
  ((< ldt (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
    (setf info -6))
  ((or (< ldvl 1) (and leftv (< ldvl n))) (setf info -8))
  ((or (< ldvr 1) (and rightv (< ldvr n))) (setf info -10))
  ((< mm m) (setf info -11)))
(cond ((/= info 0)
  (xerbla "ZTREVC" (f2cl-lib:int-sub info)) (go end_label)))
(if (= n 0) (go end_label)) (setf unfl (dlamch "Safe minimum"))
(setf ovfl (/ one unfl))
(multiple-value-bind (var-0 var-1)
  (dlabab unfl ovfl) (declare (ignore))
  (when var-0 (setf unfl var-0)) (when var-1 (setf ovfl var-1)))
(setf ulp (dlamch "Precision")) (setf smlnum (* unfl (/ n ulp)))
(f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))

```

```

(> i n) nil)
(tagbody
  (setf
    (f2cl-lib:fref work-%data%
      ((f2cl-lib:int-add i n)) ((1 *)) work-%offset%)
    (f2cl-lib:fref t$-%data% (i i)
      ((1 ldt) (1 *)) t$-%offset%))
    label20))
(setf (f2cl-lib:fref rwork-%data% (1) ((1 *)) rwork-%offset%) zero)
(f2cl-lib:fdo (j 2 (f2cl-lib:int-add j 1))
  (> j n) nil)
(tagbody
  (setf (f2cl-lib:fref rwork-%data% (j)
    ((1 *)) rwork-%offset%)
    (dzasum (f2cl-lib:int-sub j 1)
      (f2cl-lib:array-slice t$-%data%
        f2cl-lib:complex16 (1 j) ((1 ldt) (1 *))
        t$-%offset%)
      1))
    label30))
(cond
  (rightv (setf is m)
    (f2cl-lib:fdo (ki n (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
      (> ki 1)
      nil)
    (tagbody
      (cond
        (somev
          (if (not (f2cl-lib:fref select-%data% (ki)
            ((1 *)) select-%offset%))
            (go label80))))
      (setf smin
        (max
          (* ulp
            (cabs1 (f2cl-lib:fref t$-%data% (ki ki)
              ((1 ldt) (1 *)) t$-%offset%))
            smlnum))
        (setf (f2cl-lib:fref work-%data% (1)
          ((1 *)) work-%offset%) cmone)
        (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
          (> k
            (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
            nil)
          (tagbody
            (setf (f2cl-lib:fref work-%data% (k)
              ((1 *)) work-%offset%)
              (- (f2cl-lib:fref t$-%data% (k ki)
                ((1 ldt) (1 *)) t$-%offset%)))
              label40))
            (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))

```



```

(> k
  (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
nil)
(tagbody
  (setf (f2cl-lib:fref t$-%data% (k k)
    ((1 ldt) (1 *)) t$-%offset%)
    (- (f2cl-lib:fref t$-%data% (k k)
      ((1 ldt) (1 *)) t$-%offset%)
      (f2cl-lib:fref t$-%data% (ki ki)
        ((1 ldt) (1 *)) t$-%offset%)))
  (if
    (< (cabs1 (f2cl-lib:fref t$-%data% (k k)
      ((1 ldt) (1 *)) t$-%offset%))
      smin)
    (setf (f2cl-lib:fref t$-%data% (k k)
      ((1 ldt) (1 *)) t$-%offset%)
      (coerce smin 'f2cl-lib:complex16)))
  label150))
(cond
  (> ki 1)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5
     var-6 var-7 var-8 var-9 var-10)
    (zlatrs "Upper" "No transpose" "Non-unit" "Y"
      (f2cl-lib:int-sub ki 1)
      t$ ldt
      (f2cl-lib:array-slice work-%data%
        f2cl-lib:complex16 (1) ((1 *))
        work-%offset%)
      scale rwork info)
    (declare (ignore var-0 var-1 var-2 var-3
      var-4 var-5 var-7 var-9))
    (setf ldt var-6)
    (setf scale var-8)
    (setf info var-10))
  (setf (f2cl-lib:fref work-%data% (ki)
    ((1 *)) work-%offset%)
    (coerce scale 'f2cl-lib:complex16)))
  (cond
    ((not over)
      (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
        (zcopy ki
          (f2cl-lib:array-slice work-%data%
            f2cl-lib:complex16 (1) ((1 *))
            work-%offset%)
          1
          (f2cl-lib:array-slice vr-%data%
            f2cl-lib:complex16 (1 is)
            ((1 ldvr) (1 *)) vr-%offset%)
          1)

```

```

(declare (ignore var-1 var-2 var-3 var-4))
(when var-0 (setf ki var-0)))
(setf ii
  (multiple-value-bind (ret-val var-0 var-1 var-2)
    (izamax ki
      (f2cl-lib:array-slice vr-%data%
        f2cl-lib:complex16 (1 is)
        ((1 ldvr) (1 *)) vr-%offset%)
      1)
    (declare (ignore var-1 var-2))
    (when var-0 (setf ki var-0)) ret-val))
(setf remax
  (/ one
    (cabs1
      (f2cl-lib:fref vr-%data% (ii is)
        ((1 ldvr) (1 *)) vr-%offset%))))
(multiple-value-bind (var-0 var-1 var-2 var-3)
  (zdscal ki remax
    (f2cl-lib:array-slice vr-%data%
      f2cl-lib:complex16 (1 is)
      ((1 ldvr) (1 *)) vr-%offset%)
    1)
  (declare (ignore var-2 var-3))
  (when var-0 (setf ki var-0))
  (when var-1 (setf remax var-1)))
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
              (f2cl-lib:int-add k 1))
  ((> k n)
    nil)
  (tagbody
    (setf (f2cl-lib:fref vr-%data% (k is)
      ((1 ldvr) (1 *)) vr-%offset%)
      cmzero)
    label60)))
(t
  (if (> ki 1)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5
       var-6 var-7 var-8 var-9 var-10)
      (zgemv "N" n (f2cl-lib:int-sub ki 1) cmone vr ldvr
        (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 (1) ((1 *))
          work-%offset%)
        1 (f2cl-lib:dcmplx scale)
        (f2cl-lib:array-slice vr-%data%
          f2cl-lib:complex16 (1 ki)
          ((1 ldvr) (1 *)) vr-%offset%)
        1)
      (declare (ignore var-0 var-2 var-4 var-6
        var-7 var-8 var-9 var-10))

```

```

      (when var-1 (setf n var-1))
      (when var-3 (setf cmone var-3))
      (when var-5 (setf ldvr var-5)))
    (setf ii
      (multiple-value-bind (ret-val var-0 var-1 var-2)
        (izamax n
          (f2cl-lib:array-slice vr-%data%
            f2cl-lib:complex16 (1 ki)
            ((1 ldvr) (1 *)) vr-%offset%)
          1)
        (declare (ignore var-1 var-2))
        (when var-0 (setf n var-0)) ret-val))
    (setf remax
      (/ one
        (cabs1
          (f2cl-lib:fref vr-%data% (ii ki)
            ((1 ldvr) (1 *)) vr-%offset%)))
      (multiple-value-bind (var-0 var-1 var-2 var-3)
        (zdscal n remax
          (f2cl-lib:array-slice vr-%data%
            f2cl-lib:complex16 (1 ki)
            ((1 ldvr) (1 *)) vr-%offset%)
          1)
        (declare (ignore var-2 var-3))
        (when var-0 (setf n var-0))
        (when var-1 (setf remax var-1))))
    (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
      ((> k
        (f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
        nil)
      (tagbody
        (setf (f2cl-lib:fref t$-%data% (k k)
          ((1 ldt) (1 *)) t$-%offset%)
          (f2cl-lib:fref work-%data%
            ((f2cl-lib:int-add k n)) ((1 *))
            work-%offset%))
          label170))
        (setf is (f2cl-lib:int-sub is 1)) label80))))
  (cond
    (leftv (setf is 1)
      (f2cl-lib:fdo (ki 1 (f2cl-lib:int-add ki 1))
        ((> ki n) nil)
        (tagbody
          (cond
            (somev
              (if (not (f2cl-lib:fref select-%data% (ki)
                ((1 *)) select-%offset%))
                (go label130))))
            (setf smin
              (max

```

```

(* ulp
  (cabs1 (f2cl-lib:fref t$-%data% (ki ki)
    ((1 ldt) (1 *)) t$-%offset%)))
  smlnum))
(setf (f2cl-lib:fref work-%data% (n) ((1 *))
  work-%offset%) cmone)
(f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
  (f2cl-lib:int-add k 1))
(> k n)
  nil)
(tagbody
  (setf (f2cl-lib:fref work-%data% (k)
    ((1 *)) work-%offset%)
    (coerce
      (-
        (f2cl-lib:dconjg
          (f2cl-lib:fref t$-%data% (ki k)
            ((1 ldt) (1 *)) t$-%offset%)))
        'f2cl-lib:complex16))
    label90))
(f2cl-lib:fdo (k
  (f2cl-lib:int-add ki 1) (f2cl-lib:int-add k 1))
(> k n)
  nil)
(tagbody
  (setf (f2cl-lib:fref t$-%data% (k k)
    ((1 ldt) (1 *)) t$-%offset%)
    (- (f2cl-lib:fref t$-%data% (k k)
      ((1 ldt) (1 *)) t$-%offset%)
      (f2cl-lib:fref t$-%data% (ki ki)
        ((1 ldt) (1 *)) t$-%offset%)))
    (if
      (< (cabs1 (f2cl-lib:fref t$-%data% (k k)
        ((1 ldt) (1 *)) t$-%offset%))
        smin)
      (setf (f2cl-lib:fref t$-%data% (k k)
        ((1 ldt) (1 *)) t$-%offset%)
        (coerce smin 'f2cl-lib:complex16)))
      label100))
(cond
  (< ki n)
  (multiple-value-bind
    (var-0 var-1 var-2 var-3 var-4 var-5
     var-6 var-7 var-8 var-9 var-10)
    (zlatrs "Upper" "Conjugate transpose" "Non-unit" "Y"
      (f2cl-lib:int-sub n ki)
      (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
        ((+ ki 1) (f2cl-lib:int-add ki 1))
        ((1 ldt) (1 *)) t$-%offset%)
      ldt

```

```

(f2cl-lib:array-slice work-%data%
 f2cl-lib:complex16 ((+ ki 1))
 ((1 *)) work-%offset%)
scale rwork info)
(declare (ignore var-0 var-1 var-2 var-3
          var-4 var-5 var-7 var-9))
(setf ldt var-6)
(setf scale var-8)
(setf info var-10))
(setf (f2cl-lib:fref work-%data% (ki)
  ((1 *)) work-%offset%)
 (coerce scale 'f2cl-lib:complex16)))
(cond
 ((not over)
  (zcopy (f2cl-lib:int-add (f2cl-lib:int-sub n ki) 1)
   (f2cl-lib:array-slice work-%data%
    f2cl-lib:complex16 (ki) ((1 *))
    work-%offset%)
   1
   (f2cl-lib:array-slice vl-%data%
    f2cl-lib:complex16 (ki is)
    ((1 ldvl) (1 *)) vl-%offset%)
   1)
  (setf ii
   (f2cl-lib:int-sub
    (f2cl-lib:int-add
     (izamax (f2cl-lib:int-add
              (f2cl-lib:int-sub n ki) 1)
              (f2cl-lib:array-slice vl-%data%
               f2cl-lib:complex16 (ki is)
               ((1 ldvl) (1 *)) vl-%offset%)
              1)
            ki)
    1))
  (setf remax
   (/ one
    (cabs1
     (f2cl-lib:fref vl-%data% (ii is)
      ((1 ldvl) (1 *)) vl-%offset%))))
  (multiple-value-bind (var-0 var-1 var-2 var-3)
    (zdscal (f2cl-lib:int-add
              (f2cl-lib:int-sub n ki) 1) remax
             (f2cl-lib:array-slice vl-%data%
              f2cl-lib:complex16 (ki is)
              ((1 ldvl) (1 *)) vl-%offset%)
             1)
    (declare (ignore var-0 var-2 var-3))
    (when var-1 (setf remax var-1)))
  (f2cl-lib:fdo (k 1 (f2cl-lib:int-add k 1))
    (> k

```

```

(f2cl-lib:int-add ki (f2cl-lib:int-sub 1)))
nil)
(tagbody
  (setf (f2cl-lib:fref vl-%data% (k is)
    ((1 ldvl) (1 *)) vl-%offset%)
    cmzero)
    label110)))
(t
  (if (< ki n)
    (multiple-value-bind
      (var-0 var-1 var-2 var-3 var-4 var-5 var-6
       var-7 var-8 var-9 var-10)
      (zgemv "N" n (f2cl-lib:int-sub n ki) cmone
        (f2cl-lib:array-slice vl-%data% f2cl-lib:complex16
          (1 (f2cl-lib:int-add ki 1))
            ((1 ldvl) (1 *)) vl-%offset%)
          ldvl
        (f2cl-lib:array-slice work-%data%
          f2cl-lib:complex16 ((+ ki 1))
            ((1 *)) work-%offset%)
          1 (f2cl-lib:dcmplx scale)
        (f2cl-lib:array-slice vl-%data%
          f2cl-lib:complex16 (1 ki)
            ((1 ldvl) (1 *)) vl-%offset%)
          1)
      (declare (ignore var-0 var-2 var-4 var-6
        var-7 var-8 var-9 var-10))
      (when var-1 (setf n var-1))
      (when var-3 (setf cmone var-3))
      (when var-5 (setf ldvl var-5))))
    (setf ii
      (multiple-value-bind (ret-val var-0 var-1 var-2)
        (izamax n
          (f2cl-lib:array-slice vl-%data%
            f2cl-lib:complex16 (1 ki)
              ((1 ldvl) (1 *)) vl-%offset%)
            1)
        (declare (ignore var-1 var-2))
        (when var-0 (setf n var-0)) ret-val))
      (setf remax
        (/ one
          (cabs1
            (f2cl-lib:fref vl-%data% (ii ki)
              ((1 ldvl) (1 *)) vl-%offset%))))))
    (multiple-value-bind (var-0 var-1 var-2 var-3)
      (zdscal n remax
        (f2cl-lib:array-slice vl-%data%
          f2cl-lib:complex16 (1 ki)
            ((1 ldvl) (1 *)) vl-%offset%)
          1)

```

```

        (declare (ignore var-2 var-3))
        (when var-0 (setf n var-0))
        (when var-1 (setf remax var-1))))
    (f2cl-lib:fdo (k (f2cl-lib:int-add ki 1)
                  (f2cl-lib:int-add k 1))
      (> k n)
      nil)
    (tagbody
      (setf (f2cl-lib:fref t$-%data% (k k)
        ((1 ldt) (1 *)) t$-%offset%)
        (f2cl-lib:fref work-%data%
          ((f2cl-lib:int-add k n)) ((1 *))
          work-%offset%))
        label120))
      (setf is (f2cl-lib:int-add is 1)) label130))))
    (go end_label) end_label
    (return
      (values side howmny nil n nil ldt nil ldvl nil
        ldvr nil m nil nil info))))
  )))

```

ztrexc LAPACK

— ztrexc.input —

```

)set break resume
)sys rm -f ztrexc.output
)spool ztrexc.output
)set message test on
)set message auto off
)clear all

```

```

)spool
)lisp (bye)

```

— ztrexc.help —

```

=====
ztrexc examples
=====
=====

```

Man Page Details

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
SUBROUTINE ZTREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, INFO )
```

```
.. Scalar Arguments ..
```

```
CHARACTER          COMPQ
```

```
INTEGER            IFST, ILST, INFO, LDQ, LDT, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16         Q( LDQ, * ), T( LDT, * )
```

```
..
```

Purpose:

ZTREXC reorders the Schur factorization of a complex matrix
 $A = Q^* T Q$, so that the diagonal element of T with row index $IFST$
 is moved to row $ILST$.

The Schur form T is reordered by a unitary similarity transformation
 $Z^* H T Z$, and optionally the matrix Q of Schur vectors is updated by
 postmultiplying it with Z .

Arguments:

[in] COMPQ

COMPQ is CHARACTER*1

= 'V': update the matrix Q of Schur vectors;

= 'N': do not update Q .

[in] N

N is INTEGER

The order of the matrix T . $N \geq 0$.

[in,out] T

T is COMPLEX*16 array, dimension (LDT,N)

On entry, the upper triangular matrix T .

On exit, the reordered upper triangular matrix.

[in] LDT

LDT is INTEGER
The leading dimension of the array T. LDT \geq max(1,N).

[in,out] Q

Q is COMPLEX*16 array, dimension (LDQ,N)
On entry, if COMPQ = 'V', the matrix Q of Schur vectors.
On exit, if COMPQ = 'V', Q has been postmultiplied by the
unitary transformation matrix Z which reorders T.
If COMPQ = 'N', Q is not referenced.

[in] LDQ

LDQ is INTEGER
The leading dimension of the array Q. LDQ \geq max(1,N).

[in] IFST

IFST is INTEGER

[in] ILST

ILST is INTEGER

Specify the reordering of the diagonal elements of T:
The element with row index IFST is moved to row ILST by a
sequence of transpositions between adjacent elements.
1 \leq IFST \leq N; 1 \leq ILST \leq N.

[out] INFO

INFO is INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— ztrexc.f —

```

* =====
*      SUBROUTINE ZTREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          COMPQ
*      INTEGER            IFST, ILST, INFO, LDQ, LDT, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16         Q( LDQ, * ), T( LDT, * )
*
*      ..
*
* =====
*
*      .. Local Scalars ..
*      LOGICAL            WANTQ
*      INTEGER            K, M1, M2, M3
*      DOUBLE PRECISION   CS
*      COMPLEX*16         SN, T11, T22, TEMP
*
*      ..
*      .. External Functions ..
*      LOGICAL            LSAME
*      EXTERNAL           LSAME
*
*      ..
*      .. External Subroutines ..
*      EXTERNAL           XERBLA, ZLARTG, ZROT
*
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC          DCONJG, MAX
*
*      ..
*      .. Executable Statements ..
*
*      Decode and test the input parameters.
*
*      INFO = 0
*      WANTQ = LSAME( COMPQ, 'V' )
*      IF( .NOT.LSAME( COMPQ, 'N' ) .AND. .NOT.WANTQ ) THEN
*         INFO = -1
*      ELSE IF( N.LT.0 ) THEN
*         INFO = -2
*      ELSE IF( LDT.LT.MAX( 1, N ) ) THEN
*         INFO = -4
*      ELSE IF( LDQ.LT.1 .OR. ( WANTQ .AND. LDQ.LT.MAX( 1, N ) ) ) THEN

```

```

        INFO = -6
    ELSE IF( IFST.LT.1 .OR. IFST.GT.N ) THEN
        INFO = -7
    ELSE IF( ILST.LT.1 .OR. ILST.GT.N ) THEN
        INFO = -8
    END IF
    IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZTREXC', -INFO )
        RETURN
    END IF
*
*   Quick return if possible
*
*   IF( N.EQ.1 .OR. IFST.EQ.ILST )
$   RETURN
*
*   IF( IFST.LT.ILST ) THEN
*
*       Move the IFST-th diagonal element forward down the diagonal.
*
*       M1 = 0
*       M2 = -1
*       M3 = 1
    ELSE
*
*       Move the IFST-th diagonal element backward up the diagonal.
*
*       M1 = -1
*       M2 = 0
*       M3 = -1
    END IF
*
    DO 10 K = IFST + M1, ILST + M2, M3
*
*       Interchange the k-th and (k+1)-th diagonal elements.
*
*       T11 = T( K, K )
*       T22 = T( K+1, K+1 )
*
*       Determine the transformation to perform the interchange.
*
*       CALL ZLARTG( T( K, K+1 ), T22-T11, CS, SN, TEMP )
*
*       Apply transformation to the matrix T.
*
*       IF( K+2.LE.N )
$       CALL ZROT( N-K-1, T( K, K+2 ), LDT, T( K+1, K+2 ), LDT, CS,
$               SN )
*       CALL ZROT( K-1, T( 1, K ), 1, T( 1, K+1 ), 1, CS,
$               DCONJG( SN ) )

```

```

*
      T( K, K ) = T22
      T( K+1, K+1 ) = T11
*
      IF( WANTQ ) THEN
*
*         Accumulate transformation in the matrix Q.
*
*         CALL ZROT( N, Q( 1, K ), 1, Q( 1, K+1 ), 1, CS,
$           DCONJG( SN ) )
      END IF
*
10 CONTINUE
*
      RETURN
*
*     End of ZTREXC
*
      END

```

— LAPACK ztrexc —

```

(defun ztrexc (compq n t$ ldt q ldq ifst ilst info)
  (declare (type (simple-array character (*)) compq)
    (type (f2cl-lib:integer4) info ilst ifst ldq ldt n)
    (type (array f2cl-lib:complex16 (*)) q t$))
  (f2cl-lib:with-multi-array-data
    ((t$ f2cl-lib:complex16 t$-%data% t$-%offset%)
     (q f2cl-lib:complex16 q-%data% q-%offset%)
     (compq character compq-%data% compq-%offset%))
    (prog
      ((sn #C(0.0d0 0.0d0)) (t11 #C(0.0d0 0.0d0)) (t22 #C(0.0d0 0.0d0))
       (temp #C(0.0d0 0.0d0)) (cs 0.0d0) (k 0) (m1 0) (m2 0)
       (m3 0) (wantq nil)
       (dconjg$ 0.0))
      (declare (type (f2cl-lib:complex16) temp t22 t11 sn)
        (type (double-float) cs)
        (type (f2cl-lib:integer4) m3 m2 m1 k) (type f2cl-lib:logical wantq)
        (type (single-float) dconjg$))
      (setf info 0)
      (setf wantq
        (multiple-value-bind (ret-val var-0 var-1) (lsame compq "V")
          (declare (ignore var-1)) (when var-0 (setf compq var-0)) ret-val))
      (cond
        ((and
          (not

```

```

      (multiple-value-bind (ret-val var-0 var-1) (lsame compq "N")
        (declare (ignore var-1)) (when var-0 (setf compq var-0)) ret-val))
      (not wantq))
      (setf info -1))
      ((< n 0) (setf info -2))
      ((< ldt (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
        (setf info -4))
      ((or (< ldq 1)
        (and wantq
          (< ldq (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n))))))
        (setf info -6))
      ((or (< ifst 1) (> ifst n)) (setf info -7))
      ((or (< ilst 1) (> ilst n)) (setf info -8)))
      (cond ((/= info 0)
        (xerbla "ZTREXC" (f2cl-lib:int-sub info)) (go end_label)))
      (if (or (= n 1) (= ifst ilst)) (go end_label))
      (cond ((< ifst ilst) (setf m1 0) (setf m2 -1) (setf m3 1))
        (t (setf m1 -1) (setf m2 0) (setf m3 -1)))
      (f2cl-lib:fdo (k (f2cl-lib:int-add ifst m1) (f2cl-lib:int-add k m3))
        ((> k
          (f2cl-lib:int-add ilst m2))
          nil)
        (tagbody
          (setf t11 (f2cl-lib:fref t$-%data% (k k)
            ((1 ldt) (1 *)) t$-%offset%))
          (setf t22
            (f2cl-lib:fref t$-%data%
              ((f2cl-lib:int-add k 1) (f2cl-lib:int-add k 1))
              ((1 ldt) (1 *)) t$-%offset%))
          (multiple-value-bind (var-0 var-1 var-2 var-3 var-4)
            (zlartg
              (f2cl-lib:fref t$-%data% (k (f2cl-lib:int-add k 1))
                ((1 ldt) (1 *)) t$-%offset%)
              (- t22 t11) cs sn temp)
            (declare (ignore var-0 var-1))
            (setf cs var-2)
            (setf sn var-3)
            (setf temp var-4))
          (if (<= (f2cl-lib:int-add k 2) n)
            (zrot (f2cl-lib:int-sub n k 1)
              (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
                (k (f2cl-lib:int-add k 2)) ((1 ldt) (1 *)) t$-%offset%)
              ldt
              (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
                ((+ k 1) (f2cl-lib:int-add k 2))
                ((1 ldt) (1 *)) t$-%offset%)
              ldt cs sn))
            (zrot (f2cl-lib:int-sub k 1)
              (f2cl-lib:array-slice t$-%data% f2cl-lib:complex16 (1 k)

```

```

      ((1 ldt) (1 *))
      t$-%offset%)
1
(f2cl-lib:array-slice t$-%data% f2cl-lib:complex16
  (1 (f2cl-lib:int-add k 1)) ((1 ldt) (1 *)) t$-%offset%)
1 cs (f2cl-lib:dconjg sn))
(setf (f2cl-lib:fref t$-%data% (k k)
      ((1 ldt) (1 *)) t$-%offset%) t22)
(setf
  (f2cl-lib:fref t$-%data% ((f2cl-lib:int-add k 1)
                          (f2cl-lib:int-add k 1))
    ((1 ldt) (1 *)) t$-%offset%)
  t11)
(cond
  (wantq
    (zrot n
      (f2cl-lib:array-slice q-%data% f2cl-lib:complex16 (1 k)
        ((1 ldq) (1 *))
        q-%offset%)
      1
      (f2cl-lib:array-slice q-%data% f2cl-lib:complex16
        (1 (f2cl-lib:int-add k 1)) ((1 ldq) (1 *)) q-%offset%)
      1 cs (f2cl-lib:dconjg sn))))
  label10))
(go end_label) end_label
(return (values compq nil nil nil nil nil nil nil info))))

```

zung2r LAPACK

— zung2r.input —

```

)set break resume
)sys rm -f zung2r.output
)spool zung2r.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zung2r.help —

```
=====
zung2r examples
=====
```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

```
=====

SUBROUTINE ZUNG2R( M, N, K, A, LDA, TAU, WORK, INFO )

.. Scalar Arguments ..
INTEGER          INFO, K, LDA, M, N
..
.. Array Arguments ..
COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
..
```

Purpose:

```
=====

ZUNG2R generates an m by n complex matrix Q with orthonormal columns,
which is defined as the first n columns of a product of k elementary
reflectors of order m
```

$$Q = H(1) H(2) \dots H(k)$$

as returned by ZGEQRF.

Arguments:

[in] M

M is INTEGER
The number of rows of the matrix Q. M >= 0.

[in] N

N is INTEGER
The number of columns of the matrix Q. M >= N >= 0.

[in] K

K is INTEGER

The number of elementary reflectors whose product defines the matrix Q. $N \geq K \geq 0$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)

On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for $i = 1, 2, \dots, k$, as returned by ZGEQRF in the first k columns of its array argument A.

On exit, the m by n matrix Q.

[in] LDA

LDA is INTEGER

The first dimension of the array A. $LDA \geq \max(1, M)$.

[in] TAU

TAU is COMPLEX*16 array, dimension (K)

TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

[out] WORK

WORK is COMPLEX*16 array, dimension (N)

[out] INFO

INFO is INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument has an illegal value

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zung2r.f —


```

* =====
*      SUBROUTINE ZUNG2R( M, N, K, A, LDA, TAU, WORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          INFO, K, LDA, M, N
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
*      ..
*
* =====
*
*      .. Parameters ..
*      COMPLEX*16       ONE, ZERO
*      PARAMETER        ( ONE = ( 1.0D+0, 0.0D+0 ),
* $                     ZERO = ( 0.0D+0, 0.0D+0 ) )
*      ..
*      .. Local Scalars ..
*      INTEGER          I, J, L
*      ..
*      .. External Subroutines ..
*      EXTERNAL         XERBLA, ZLARF, ZSCAL
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC        MAX
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
*      INFO = 0
*      IF( M.LT.0 ) THEN
*          INFO = -1
*      ELSE IF( N.LT.0 .OR. N.GT.M ) THEN
*          INFO = -2
*      ELSE IF( K.LT.0 .OR. K.GT.N ) THEN
*          INFO = -3
*      ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
*          INFO = -5
*      END IF
*      IF( INFO.NE.0 ) THEN
*          CALL XERBLA( 'ZUNG2R', -INFO )
*          RETURN
*      END IF
*
*

```

```

*      Quick return if possible
*
      IF( N.LE.0 )
$      RETURN
*
*      Initialise columns k+1:n to columns of the unit matrix
*
      DO 20 J = K + 1, N
        DO 10 L = 1, M
          A( L, J ) = ZERO
10      CONTINUE
        A( J, J ) = ONE
20      CONTINUE
*
      DO 40 I = K, 1, -1
*
*      Apply H(i) to A(i:m,i:n) from the left
*
      IF( I.LT.N ) THEN
        A( I, I ) = ONE
        CALL ZLARF( 'Left', M-I+1, N-I, A( I, I ), 1, TAU( I ),
$          A( I, I+1 ), LDA, WORK )
      END IF
      IF( I.LT.M )
$      CALL ZSCAL( M-I, -TAU( I ), A( I+1, I ), 1 )
      A( I, I ) = ONE - TAU( I )
*
*      Set A(1:i-1,i) to zero
*
      DO 30 L = 1, I - 1
        A( L, I ) = ZERO
30      CONTINUE
40      CONTINUE
      RETURN
*
*      End of ZUNG2R
*
      END

```

— LAPACK zung2r —

```

(let*
  ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16))
   (zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (type (f2cl-lib:complex16) zero)
   (ignorable one zero))

```

```

(defun zung2r (m n k a lda tau work info)
  (declare (type (f2cl-lib:integer4) info lda k n m)
    (type (array f2cl-lib:complex16 (*)) work tau a))
  (f2cl-lib:with-multi-array-data
    ((a f2cl-lib:complex16 a-%data% a-%offset%)
     (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
     (work f2cl-lib:complex16 work-%data% work-%offset%))
    (prog ((i 0) (j 0) (l 0))
      (declare (type (f2cl-lib:integer4) l j i)) (setf info 0)
      (cond ((< m 0) (setf info -1)) ((or (< n 0) (> n m)) (setf info -2))
        ((or (< k 0) (> k n)) (setf info -3))
        ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m)))
         (setf info -5)))
      (cond (/= info 0)
        (xerbla "ZUNG2R" (f2cl-lib:int-sub info)) (go end_label)))
    (if (<= n 0) (go end_label))
    (f2cl-lib:fdo (j (f2cl-lib:int-add k 1) (f2cl-lib:int-add j 1))
      ((> j n) nil)
      (tagbody
        (f2cl-lib:fdo (l 1 (f2cl-lib:int-add 1 1))
          ((> l m) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data% (l j)
              ((1 lda) (1 *)) a-%offset%) zero)
              label10))
            (setf (f2cl-lib:fref a-%data% (j j)
              ((1 lda) (1 *)) a-%offset%) one) label20)
          )
    (f2cl-lib:fdo (i k (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
      ((> i 1) nil)
      (tagbody
        (cond
          ((< i n)
            (setf (f2cl-lib:fref a-%data% (i i)
              ((1 lda) (1 *)) a-%offset%) one)
            (multiple-value-bind
              (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
              (zlarf "Left" (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
                (f2cl-lib:int-sub n i)
                (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
                  (i i) ((1 lda) (1 *))
                  a-%offset%)
                1 (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)
                (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
                  (i (f2cl-lib:int-add i 1)) ((1 lda) (1 *)) a-%offset%)
                lda work)
              (declare (ignore var-0 var-1 var-2 var-3
                var-4 var-5 var-6 var-8))
              (setf lda var-7))))
          (if (< i m)

```

```

(zscal (f2cl-lib:int-sub m i)
  (- (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%))
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    ((+ i 1) i)
    ((1 lda) (1 *)) a-%offset%
    1))
(setf (f2cl-lib:fref a-%data% (i i)
  ((1 lda) (1 *)) a-%offset%
  (- one (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
  (f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1
    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    nil)
  (tagbody
    (setf (f2cl-lib:fref a-%data% (1 i)
      ((1 lda) (1 *)) a-%offset% zero)
      label30))
    label40))
  (go end_label)
end_label (return (values nil nil nil nil lda nil nil info)))
)))

```

—————

zunghr LAPACK

— zunghr.input —

```

)set break resume
)sys rm -f zunghr.output
)spool zunghr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

—————

— zunghr.help —

```

=====
zunghr examples
=====

```

```
=====
Man Page Details
=====
```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

```
Definition:
=====
```

```
SUBROUTINE ZUNGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
```

```
.. Scalar Arguments ..
```

```
INTEGER          IHI, ILO, INFO, LDA, LWORK, N
```

```
..
```

```
.. Array Arguments ..
```

```
COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
```

```
..
```

```
Purpose:
=====
```

ZUNGHR generates a complex unitary matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N, as returned by ZGEHRD:

$$Q = H(i\text{lo}) H(i\text{lo}+1) \dots H(i\text{hi}-1).$$

```
Arguments:
=====
```

```
[in] N
```

N is INTEGER

The order of the matrix Q. N >= 0.

```
[in] ILO
```

ILO is INTEGER

```
[in] IHI
```

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of ZGEHRD. Q is equal to the unit matrix except in the submatrix Q(i\text{lo}+1:i\text{hi},i\text{lo}+1:i\text{hi}).

1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)
On entry, the vectors which define the elementary reflectors,
as returned by ZGEHRD.
On exit, the N-by-N unitary matrix Q.

[in] LDA

LDA is INTEGER
The leading dimension of the array A. LDA \geq max(1,N).

[in] TAU

TAU is COMPLEX*16 array, dimension (N-1)
TAU(i) must contain the scalar factor of the elementary
reflector H(i), as returned by ZGEHRD.

[out] WORK

WORK is COMPLEX*16 array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER
The dimension of the array WORK. LWORK \geq IHI-ILO.
For optimum performance LWORK \geq (IHI-ILO)*NB, where NB is
the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

[out] INFO

INFO is INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

Authors:

=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zunghr.f —

```

* =====
*      SUBROUTINE ZUNGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,      --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      INTEGER          IHI, ILO, INFO, LDA, LWORK, N
*      ..
*      .. Array Arguments ..
*      COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
*      ..
*
* =====
*
*      .. Parameters ..
*      COMPLEX*16       ZERO, ONE
*      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ),
*      $                ONE = ( 1.0D+0, 0.0D+0 ) )
*      ..
*      .. Local Scalars ..
*      LOGICAL          LQUERY
*      INTEGER          I, IINFO, J, LWKOPT, NB, NH
*      ..
*      .. External Subroutines ..
*      EXTERNAL         XERBLA, ZUNGQR
*      ..
*      .. External Functions ..
*      INTEGER          ILAENV
*      EXTERNAL         ILAENV
*      ..
*      .. Intrinsic Functions ..
*      INTRINSIC        MAX, MIN
*      ..
*      .. Executable Statements ..
*
*      Test the input arguments
*
*      INFO = 0
*      NH = IHI - ILO
*      LQUERY = ( LWORK.EQ.-1 )

```

```

      IF( N.LT.0 ) THEN
        INFO = -1
      ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, N ) ) THEN
        INFO = -2
      ELSE IF( IHI.LT.MIN( ILO, N ) .OR. IHI.GT.N ) THEN
        INFO = -3
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
        INFO = -5
      ELSE IF( LWORK.LT.MAX( 1, NH ) .AND. .NOT.LQUERY ) THEN
        INFO = -8
      END IF
*
      IF( INFO.EQ.0 ) THEN
        NB = ILAENV( 1, 'ZUNGQR', ' ', NH, NH, NH, -1 )
        LWKOPT = MAX( 1, NH )*NB
        WORK( 1 ) = LWKOPT
      END IF
*
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZUNGHR', -INFO )
        RETURN
      ELSE IF( LQUERY ) THEN
        RETURN
      END IF
*
*      Quick return if possible
*
      IF( N.EQ.0 ) THEN
        WORK( 1 ) = 1
        RETURN
      END IF
*
*      Shift the vectors which define the elementary reflectors one
*      column to the right, and set the first ilo and the last n-ihl
*      rows and columns to those of the unit matrix
*
      DO 40 J = IHI, ILO + 1, -1
        DO 10 I = 1, J - 1
          A( I, J ) = ZERO
10      CONTINUE
        DO 20 I = J + 1, IHI
          A( I, J ) = A( I, J-1 )
20      CONTINUE
        DO 30 I = IHI + 1, N
          A( I, J ) = ZERO
30      CONTINUE
40      CONTINUE
      DO 60 J = 1, ILO
        DO 50 I = 1, N
          A( I, J ) = ZERO

```



```

50    CONTINUE
      A( J, J ) = ONE
60 CONTINUE
      DO 80 J = IHI + 1, N
        DO 70 I = 1, N
          A( I, J ) = ZERO
70    CONTINUE
        A( J, J ) = ONE
80 CONTINUE
*
      IF( NH.GT.0 ) THEN
*
*        Generate Q(ilo+1:ihi,ilo+1:ihi)
*
          CALL ZUNGQR( NH, NH, NH, A( ILO+1, ILO+1 ), LDA, TAU( ILO ),
$              WORK, LWORK, IINFO )
        END IF
        WORK( 1 ) = LWKOPT
        RETURN
*
*    End of ZUNGHR
*
      END

```

— LAPACK zunghr —

```

(let*
  ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16))
   (one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) zero) (type (f2cl-lib:complex16) one)
    (ignorable zero one))
  (defun zunghr (n ilo ihi a lda tau work lwork info)
    (declare (type (f2cl-lib:integer4) info lwork lda ihi ilo n)
      (type (array f2cl-lib:complex16 (*)) work tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (prog
        ((i 0) (iinfo 0) (j 0) (lwkopt 0) (nb 0) (nh 0) (lquery nil))
        (declare (type (f2cl-lib:integer4) nh nb lwkopt j iinfo i)
          (type f2cl-lib:logical lquery))
        (setf info 0) (setf nh (f2cl-lib:int-sub ihi ilo))
        (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical))
        (cond ((< n 0) (setf info -1))
              ((or (< ilo 1)

```

```

(> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
(setf info -2))
((or (< ihi (min (the f2cl-lib:integer4 ilo)
                (the f2cl-lib:integer4 n)))
    (> ihi n))
 (setf info -3))
(< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 n)))
(setf info -5))
((and (< lwork (max (the f2cl-lib:integer4 1)
                    (the f2cl-lib:integer4 nh)))
    (not lquery))
 (setf info -8)))
(cond
 ((= info 0)
  (setf nb
    (multiple-value-bind (ret-val var-0 var-1 var-2 var-3
                        var-4 var-5 var-6)
      (ilaenv 1 "ZUNGQR" " " nh nh nh -1)
      (declare (ignore var-0 var-1 var-2 var-6))
      (when var-3 (setf nh var-3))
      (when var-4 (setf nh var-4))
      (when var-5 (setf nh var-5)) ret-val))
    (setf lwkopt
      (f2cl-lib:int-mul
        (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nh)) nb))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
      (coerce lwkopt 'f2cl-lib:complex16))))
 (cond (/= info 0)
  (xerbla "ZUNGHR" (f2cl-lib:int-sub info)) (go end_label))
 (lquery (go end_label)))
(cond
 ((= n 0)
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
    (coerce 1 'f2cl-lib:complex16))
  (go end_label)))
(f2cl-lib:fdo (j ihi (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
  ((> j
    (f2cl-lib:int-add ilo 1))
   nil)
 (tagbody
  (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
    ((> i
      (f2cl-lib:int-add j (f2cl-lib:int-sub 1)))
     nil)
   (tagbody
    (setf (f2cl-lib:fref a-%data% (i j)
                        ((1 lda) (1 *)) a-%offset%) zero)
    label10))
  (f2cl-lib:fdo (i (f2cl-lib:int-add j 1)
                (f2cl-lib:int-add i 1))

```

```

(> i ihi)
  nil)
  (tagbody
    (setf (f2cl-lib:fref a-%data% (i j)
      ((1 lda) (1 *)) a-%offset%)
      (f2cl-lib:fref a-%data% (i (f2cl-lib:int-sub j 1))
        ((1 lda) (1 *))
        a-%offset%))
      label20))
    (f2cl-lib:fdo (i (f2cl-lib:int-add ihi 1)
      (f2cl-lib:int-add i 1))
      (> i n)
      nil)
    (tagbody
      (setf (f2cl-lib:fref a-%data% (i j)
        ((1 lda) (1 *)) a-%offset%) zero)
        label30))
      label40))
(f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
  (> j ilo) nil)
  (tagbody
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
      (> i n) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%) zero)
          label50))
        (setf (f2cl-lib:fref a-%data% (j j)
          ((1 lda) (1 *)) a-%offset%) one) label60)
        )
    (f2cl-lib:fdo (j (f2cl-lib:int-add ihi 1) (f2cl-lib:int-add j 1))
      (> j n) nil)
      (tagbody
        (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
          (> i n) nil)
          (tagbody
            (setf (f2cl-lib:fref a-%data% (i j)
              ((1 lda) (1 *)) a-%offset%) zero)
              label70))
            (setf (f2cl-lib:fref a-%data% (j j)
              ((1 lda) (1 *)) a-%offset%) one) label80)
            )
        )
    )
  (cond
    (> nh 0)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5
      var-6 var-7 var-8)
      (zungqr nh nh nh
        (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
          ((+ ilo 1) (f2cl-lib:int-add ilo 1)) ((1 lda) (1 *)) a-%offset%)
          lda

```

```

(f2cl-lib:array-slice tau-%data% f2cl-lib:complex16 (ilo) ((1 *))
  tau-%offset%)
work lwork iinfo)
(declare (ignore var-3 var-5 var-6)) (when var-0 (setf nh var-0))
(when var-1 (setf nh var-1)) (when var-2 (setf nh var-2))
(when var-4 (setf lda var-4)) (when var-7 (setf lwork var-7))
(when var-8 (setf iinfo var-8))))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce lwkopt 'f2cl-lib:complex16))
(go end_label) end_label
(return (values nil nil nil nil lda nil nil lwork info))))

```

zungqr LAPACK

— zungqr.input —

```

)set break resume
)sys rm -f zungqr.output
)spool zungqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zungqr.help —

```

=====
zungqr examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```
SUBROUTINE ZUNGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
```

```
.. Scalar Arguments ..
INTEGER          INFO, K, LDA, LWORK, M, N
..
.. Array Arguments ..
COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
..
```

Purpose:

=====

ZUNGQR generates an M-by-N complex matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

$$Q = H(1) H(2) \dots H(k)$$

as returned by ZGEQRF.

Arguments:

=====

[in] M

M is INTEGER
The number of rows of the matrix Q. $M \geq 0$.

[in] N

N is INTEGER
The number of columns of the matrix Q. $M \geq N \geq 0$.

[in] K

K is INTEGER
The number of elementary reflectors whose product defines the matrix Q. $N \geq K \geq 0$.

[in,out] A

A is COMPLEX*16 array, dimension (LDA,N)
On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for $i = 1, 2, \dots, k$, as returned by ZGEQRF in the first k columns of its array argument A.
On exit, the M-by-N matrix Q.

[in] LDA

LDA is INTEGER

The first dimension of the array A. $LDA \geq \max(1, M)$.

[in] TAU

TAU is COMPLEX*16 array, dimension (K)

TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

[out] WORK

WORK is COMPLEX*16 array, dimension (MAX(1, LWORK))

On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER

The dimension of the array WORK. $LWORK \geq \max(1, N)$.

For optimum performance $LWORK \geq N \cdot NB$, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

[out] INFO

INFO is INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument has an illegal value

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zungqr.f —

* =====

```

      SUBROUTINE ZUNGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*   November 2011
*
*   .. Scalar Arguments ..
      INTEGER          INFO, K, LDA, LWORK, M, N
*
*   ..
*
*   .. Array Arguments ..
      COMPLEX*16       A( LDA, * ), TAU( * ), WORK( * )
*
*   ..
*
* =====
*
*   .. Parameters ..
      COMPLEX*16       ZERO
      PARAMETER        ( ZERO = ( 0.0D+0, 0.0D+0 ) )
*
*   ..
*
*   .. Local Scalars ..
      LOGICAL          LQUERY
      INTEGER          I, IB, IINFO, IWS, J, KI, KK, L, LDWORK,
$                      LWKOPT, NB, NBMIN, NX
*
*   ..
*
*   .. External Subroutines ..
      EXTERNAL          XERBLA, ZLARFB, ZLARFT, ZUNG2R
*
*   ..
*
*   .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*
*   ..
*
*   .. External Functions ..
      INTEGER          ILAENV
      EXTERNAL          ILAENV
*
*   ..
*
*   .. Executable Statements ..
*
*   Test the input arguments
*
      INFO = 0
      NB = ILAENV( 1, 'ZUNGQR', ' ', M, N, K, -1 )
      LWKOPT = MAX( 1, N )*NB
      WORK( 1 ) = LWKOPT
      LQUERY = ( LWORK.EQ.-1 )
      IF( M.LT.0 ) THEN
         INFO = -1
      ELSE IF( N.LT.0 .OR. N.GT.M ) THEN
         INFO = -2
      ELSE IF( K.LT.0 .OR. K.GT.N ) THEN
         INFO = -3

```

```

ELSE IF( LDA.LT.MAX( 1, M ) ) THEN
    INFO = -5
ELSE IF( LWORK.LT.MAX( 1, N ) .AND. .NOT.LQUERY ) THEN
    INFO = -8
END IF
IF( INFO.NE.0 ) THEN
    CALL XERBLA( 'ZUNGQR', -INFO )
    RETURN
ELSE IF( LQUERY ) THEN
    RETURN
END IF

*
* Quick return if possible
*
IF( N.LE.0 ) THEN
    WORK( 1 ) = 1
    RETURN
END IF

*
NBMIN = 2
NX = 0
IWS = N
IF( NB.GT.1 .AND. NB.LT.K ) THEN

*
* Determine when to cross over from blocked to unblocked code.
*
NX = MAX( 0, ILAENV( 3, 'ZUNGQR', ' ', M, N, K, -1 ) )
IF( NX.LT.K ) THEN

*
* Determine if workspace is large enough for blocked code.
*
LDWORK = N
IWS = LDWORK*NB
IF( LWORK.LT.IWS ) THEN

*
* Not enough workspace to use optimal NB: reduce NB and
* determine the minimum value of NB.
*
NB = LWORK / LDWORK
NBMIN = MAX( 2, ILAENV( 2, 'ZUNGQR', ' ', M, N, K, -1 ) )
END IF
END IF
END IF

*
IF( NB.GE.NBMIN .AND. NB.LT.K .AND. NX.LT.K ) THEN

*
* Use blocked code after the last block.
* The first kk columns are handled by the block method.
*
KI = ( ( K-NX-1 ) / NB ) * NB

```



```

        KK = MIN( K, KI+NB )
*
*      Set A(1:kk,kk+1:n) to zero.
*
        DO 20 J = KK + 1, N
            DO 10 I = 1, KK
                A( I, J ) = ZERO
10          CONTINUE
20          CONTINUE
        ELSE
            KK = 0
        END IF
*
*      Use unblocked code for the last or only block.
*
        IF( KK.LT.N )
$      CALL ZUNG2R( M-KK, N-KK, K-KK, A( KK+1, KK+1 ), LDA,
$                  TAU( KK+1 ), WORK, IINFO )
*
        IF( KK.GT.0 ) THEN
*
*      Use blocked code
*
        DO 50 I = KI + 1, 1, -NB
            IB = MIN( NB, K-I+1 )
            IF( I+IB.LE.N ) THEN
*
*      Form the triangular factor of the block reflector
*      H = H(i) H(i+1) . . . H(i+ib-1)
*
                CALL ZLARFT( 'Forward', 'Columnwise', M-I+1, IB,
$                  A( I, I ), LDA, TAU( I ), WORK, LDWORK )
*
*      Apply H to A(i:m,i+ib:n) from the left
*
                CALL ZLARFB( 'Left', 'No transpose', 'Forward',
$                  'Columnwise', M-I+1, N-I-IB+1, IB,
$                  A( I, I ), LDA, WORK, LDWORK, A( I, I+IB ),
$                  LDA, WORK( IB+1 ), LDWORK )
                END IF
*
*      Apply H to rows i:m of current block
*
                CALL ZUNG2R( M-I+1, IB, IB, A( I, I ), LDA, TAU( I ), WORK,
$                  IINFO )
*
*      Set rows 1:i-1 of current block to zero
*
                DO 40 J = I, I + IB - 1
                    DO 30 L = 1, I - 1

```

```

          A( L, J ) = ZERO
30      CONTINUE
40      CONTINUE
50      CONTINUE
      END IF
*
      WORK( 1 ) = IWS
      RETURN
*
*      End of ZUNGQR
*
      END

```

— LAPACK zungqr —

```

(let* ((zero (coerce (f2cl-lib:cmplx 0.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) zero) (ignorable zero))
  (defun zungqr (m n k a lda tau work lwork info)
    (declare (type (f2cl-lib:integer4) info lwork lda k n m)
      (type (array f2cl-lib:complex16 (*)) work tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%))
      (prog
        ((i 0) (ib 0) (iinfo 0) (iws 0) (j 0) (ki 0) (kk 0) (l 0) (ldwork 0)
         (lwkopt 0) (nb 0) (nbmin 0) (nx 0) (lquery nil))
        (declare
          (type (f2cl-lib:integer4) nx nbmin nb lwkopt ldwork
            l kk ki j iws iinfo ib i)
          (type f2cl-lib:logical lquery))
        (setf info 0)
        (setf nb
          (multiple-value-bind (ret-val var-0 var-1 var-2 var-3
                                var-4 var-5 var-6)
            (ilaenv 1 "ZUNGQR" " " m n k -1)
            (declare (ignore var-0 var-1 var-2 var-6))
            (when var-3 (setf m var-3)) (when var-4 (setf n var-4))
            (when var-5 (setf k var-5)) ret-val)))
        (setf lwkopt
          (f2cl-lib:int-mul (max (the f2cl-lib:integer4 1)
                                (the f2cl-lib:integer4 n))
            nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce lwkopt 'f2cl-lib:complex16))
        (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical)))

```

```

(cond ((< m 0) (setf info -1)) ((or (< n 0) (> n m)) (setf info -2))
      ((or (< k 0) (> k n)) (setf info -3))
      ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m)))
        (setf info -5))
      ((and (< lwork (max (the f2cl-lib:integer4 1)
                          (the f2cl-lib:integer4 n)))
            (not lquery))
        (setf info -8)))
(cond ((/= info 0)
      (xerbla "ZUNGQR" (f2cl-lib:int-sub info)) (go end_label))
      (lquery (go end_label)))
(cond
  ((<= n 0)
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce 1 'f2cl-lib:complex16))
   (go end_label)))
(setf nbmin 2) (setf nx 0) (setf iws n)
(cond
  ((and (> nb 1) (< nb k))
   (setf nx
        (max (the f2cl-lib:integer4 0)
              (the f2cl-lib:integer4
                  (multiple-value-bind (ret-val var-0 var-1 var-2
                                        var-3 var-4 var-5 var-6)
                    (ilaenv 3 "ZUNGQR" " " m n k -1)
                    (declare (ignore var-0 var-1 var-2 var-6))
                    (when var-3 (setf m var-3))
                    (when var-4 (setf n var-4))
                    (when var-5 (setf k var-5)) ret-val))))))
   (cond
     ((< nx k) (setf ldwork n) (setf iws (f2cl-lib:int-mul ldwork nb))
      (cond
        ((< lwork iws)
         (setf nb (the f2cl-lib:integer4 (truncate lwork ldwork)))
         (setf nbmin
              (max (the f2cl-lib:integer4 2)
                    (the f2cl-lib:integer4
                        (multiple-value-bind
                          (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                          (ilaenv 2 "ZUNGQR" " " m n k -1)
                          (declare (ignore var-0 var-1 var-2 var-6))
                          (when var-3 (setf m var-3)) (when var-4 (setf n var-4))
                          (when var-5 (setf k var-5)) ret-val))))))))))
     (t (t))))
   (cond
     ((and (>= nb nbmin) (< nb k) (< nx k))
      (setf ki (* (the f2cl-lib:integer4 (truncate (- k nx 1) nb)) nb))
      (setf kk
            (min (the f2cl-lib:integer4 k)
                  (the f2cl-lib:integer4 (f2cl-lib:int-add ki nb))))
      (f2cl-lib:fdo (j (f2cl-lib:int-add kk 1) (f2cl-lib:int-add j 1))

```

```

      (> j n)
      nil)
    (tagbody
      (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
        (> i kk) nil)
      (tagbody
        (setf (f2cl-lib:fref a-%data% (i j)
          ((1 lda) (1 *)) a-%offset%) zero)
          label10))
      label20)))
    (t (setf kk 0)))
  (if (< kk n)
    (multiple-value-bind (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7)
      (zung2r (f2cl-lib:int-sub m kk) (f2cl-lib:int-sub n kk)
        (f2cl-lib:int-sub k kk)
        (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
          ((+ kk 1) (f2cl-lib:int-add kk 1)) ((1 lda) (1 *)) a-%offset%)
        lda
        (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16
          ((+ kk 1)) ((1 *))
          tau-%offset%)
        work iinfo)
      (declare (ignore var-0 var-1 var-2 var-3 var-5 var-6))
      (setf lda var-4)
      (setf iinfo var-7)))
  (cond
    (> kk 0)
    (f2cl-lib:fdo (i (f2cl-lib:int-add ki 1)
      (f2cl-lib:int-add i (f2cl-lib:int-sub nb)))
      (> i 1) nil)
    (tagbody
      (setf ib
        (min (the f2cl-lib:integer4 nb)
          (the f2cl-lib:integer4
            (f2cl-lib:int-add (f2cl-lib:int-sub k i) 1)))))
    (cond
      (<= (f2cl-lib:int-add i ib) n)
      (multiple-value-bind
        (var-0 var-1 var-2 var-3 var-4 var-5
          var-6 var-7 var-8)
        (zlarft "Forward" "Columnwise"
          (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1) ib
          (f2cl-lib:array-slice a-%data%
            f2cl-lib:complex16 (i i) ((1 lda) (1 *))
            a-%offset%)
          lda
          (f2cl-lib:array-slice tau-%data%
            f2cl-lib:complex16 (i) ((1 *))
            tau-%offset%)
          work ldwork)

```

```

(declare (ignore var-0 var-1 var-2 var-3
              var-4 var-6 var-7))
(setf lda var-5) (setf ldwork var-8))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4
   var-5 var-6 var-7 var-8 var-9 var-10
   var-11 var-12 var-13 var-14)
  (zlarfb "Left" "No transpose" "Forward" "Columnwise"
    (f2cl-lib:int-add (f2cl-lib:int-sub m i) 1)
    (f2cl-lib:int-add (f2cl-lib:int-sub n i ib) 1) ib
    (f2cl-lib:array-slice a-%data%
      f2cl-lib:complex16 (i i) ((1 lda) (1 *)))
    a-%offset%)
  lda work ldwork
  (f2cl-lib:array-slice a-%data% f2cl-lib:complex16
    (i (f2cl-lib:int-add i ib))
    ((1 lda) (1 *)) a-%offset%)
  lda
  (f2cl-lib:array-slice work-%data%
    f2cl-lib:complex16 ((+ ib 1)) ((1 *)))
  work-%offset%)
  ldwork)
(declare
  (ignore var-0 var-1 var-2 var-3 var-4
            var-5 var-7 var-9 var-11 var-13))
(setf ib var-6)
(setf lda var-8)
(setf ldwork var-10)
(setf lda var-12)
(setf ldwork var-14))))
(multiple-value-bind (var-0 var-1 var-2 var-3
                      var-4 var-5 var-6 var-7)
  (zung2r (f2cl-lib:int-add
    (f2cl-lib:int-sub m i) 1) ib ib
    (f2cl-lib:array-slice a-%data%
      f2cl-lib:complex16 (i i) ((1 lda) (1 *)))
    a-%offset%)
  lda
  (f2cl-lib:array-slice tau-%data%
    f2cl-lib:complex16 (i) ((1 *)))
  tau-%offset%)
  work iinfo)
(declare (ignore var-0 var-1 var-2
              var-3 var-5 var-6))
(setf lda var-4)
(setf iinfo var-7))
(f2cl-lib:fdo (j i (f2cl-lib:int-add j 1))
  ((> j
    (f2cl-lib:int-add i ib (f2cl-lib:int-sub 1)))
  nil)

```

```

(tagbody
  (f2cl-lib:fdo (1 1 (f2cl-lib:int-add 1 1))
  (> 1
    (f2cl-lib:int-add i (f2cl-lib:int-sub 1)))
    nil)
  (tagbody
    (setf (f2cl-lib:fref a-%data% (1 j)
      ((1 lda) (1 *)) a-%offset%) zero)
      label30))
      label40))
      label150))))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce iws 'f2cl-lib:complex16))
  (go end_label)
end_label
  (return (values m n k nil lda nil nil nil info)))
)))

```

zunm2r LAPACK

— zunm2r.input —

```

)set break resume
)sys rm -f zunm2r.output
)spool zunm2r.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zunm2r.help —

```

=====
zunm2r examples
=====

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```

SUBROUTINE ZUNM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
                  WORK, INFO )

  .. Scalar Arguments ..
  CHARACTER          SIDE, TRANS
  INTEGER            INFO, K, LDA, LDC, M, N
  ..
  .. Array Arguments ..
  COMPLEX*16         A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
  ..

```

Purpose:

=====

ZUNM2R overwrites the general complex m-by-n matrix C with

```

Q * C  if SIDE = 'L' and TRANS = 'N', or

Q**H* C  if SIDE = 'L' and TRANS = 'C', or

C * Q  if SIDE = 'R' and TRANS = 'N', or

C * Q**H if SIDE = 'R' and TRANS = 'C',

```

where Q is a complex unitary matrix defined as the product of k elementary reflectors

$$Q = H(1) H(2) \dots H(k)$$

as returned by ZGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments:

=====

[in] SIDE

```

SIDE is CHARACTER*1
= 'L': apply Q or Q**H from the Left
= 'R': apply Q or Q**H from the Right

```

[in] TRANS

TRANS is CHARACTER*1
= 'N': apply Q (No transpose)
= 'C': apply Q**H (Conjugate transpose)

[in] M

M is INTEGER
The number of rows of the matrix C. $M \geq 0$.

[in] N

N is INTEGER
The number of columns of the matrix C. $N \geq 0$.

[in] K

K is INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', $M \geq K \geq 0$;
if SIDE = 'R', $N \geq K \geq 0$.

[in] A

A is COMPLEX*16 array, dimension (LDA,K)
The i-th column must contain the vector which defines the elementary reflector H(i), for $i = 1, 2, \dots, k$, as returned by ZGEQRF in the first k columns of its array argument A.
A is modified by the routine but restored on exit.

[in] LDA

LDA is INTEGER
The leading dimension of the array A.
If SIDE = 'L', $LDA \geq \max(1, M)$;
if SIDE = 'R', $LDA \geq \max(1, N)$.

[in] TAU

TAU is COMPLEX*16 array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by ZGEQRF.

[in,out] C

C is COMPLEX*16 array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q^*C or $Q^{**H}C$ or CQ^{**H} or CQ .

[in] LDC

LDC is INTEGER
 The leading dimension of the array C. $LDC \geq \max(1, M)$.

[out] WORK

WORK is COMPLEX*16 array, dimension
 (N) if SIDE = 'L',
 (M) if SIDE = 'R'

[out] INFO

INFO is INTEGER
 = 0: successful exit
 < 0: if INFO = -i, the i-th argument had an illegal value

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zunm2r.f —

```
* =====
*      SUBROUTINE ZUNM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
*      $                  WORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee,    --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER            SIDE, TRANS
*      INTEGER              INFO, K, LDA, LDC, M, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16           A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*      ..
*
*      =====
```

```

*
* .. Parameters ..
COMPLEX*16      ONE
PARAMETER      ( ONE = ( 1.0D+0, 0.0D+0 ) )
*
* ..
* .. Local Scalars ..
LOGICAL         LEFT, NOTRAN
INTEGER         I, I1, I2, I3, IC, JC, MI, NI, NQ
COMPLEX*16      AII, TAU1
*
* ..
* .. External Functions ..
LOGICAL         LSAME
EXTERNAL        LSAME
*
* ..
* .. External Subroutines ..
EXTERNAL        XERBLA, ZLARF
*
* ..
* .. Intrinsic Functions ..
INTRINSIC       DCONJG, MAX
*
* ..
* .. Executable Statements ..
*
* Test the input arguments
*
*
INFO = 0
LEFT = LSAME( SIDE, 'L' )
NOTRAN = LSAME( TRANS, 'N' )
*
* NQ is the order of Q
*
*
IF( LEFT ) THEN
    NQ = M
ELSE
    NQ = N
END IF
IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
    INFO = -1
ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'C' ) ) THEN
    INFO = -2
ELSE IF( M.LT.0 ) THEN
    INFO = -3
ELSE IF( N.LT.0 ) THEN
    INFO = -4
ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
    INFO = -5
ELSE IF( LDA.LT.MAX( 1, NQ ) ) THEN
    INFO = -7
ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
    INFO = -10
END IF

```

```

      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZUNM2R', -INFO )
        RETURN
      END IF
*
*   Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 )
$    RETURN
*
      IF( ( LEFT .AND. .NOT.NOTRAN .OR. .NOT.LEFT .AND. NOTRAN ) ) THEN
        I1 = 1
        I2 = K
        I3 = 1
      ELSE
        I1 = K
        I2 = 1
        I3 = -1
      END IF
*
      IF( LEFT ) THEN
        NI = N
        JC = 1
      ELSE
        MI = M
        IC = 1
      END IF
*
      DO 10 I = I1, I2, I3
        IF( LEFT ) THEN
*
*           H(i) or H(i)**H is applied to C(i:m,1:n)
*
          MI = M - I + 1
          IC = I
        ELSE
*
*           H(i) or H(i)**H is applied to C(1:m,i:n)
*
          NI = N - I + 1
          JC = I
        END IF
*
*       Apply H(i) or H(i)**H
*
      IF( NOTRAN ) THEN
        TAU = TAU( I )
      ELSE
        TAU = DCONJG( TAU( I ) )
      END IF

```

```

      AII = A( I, I )
      A( I, I ) = ONE
      CALL ZLARF( SIDE, MI, NI, A( I, I ), 1, TAUI, C( IC, JC ), LDC,
$          WORK )
      A( I, I ) = AII
10 CONTINUE
      RETURN
*
*      End of ZUNM2R
*
      END

```

— LAPACK zunm2r —

```

(let* ((one (coerce (f2cl-lib:cmplx 1.0d0 0.0d0) 'f2cl-lib:complex16)))
  (declare (type (f2cl-lib:complex16) one) (ignorable one))
  (defun zunm2r (side trans m n k a lda tau c ldc work info)
    (declare (type (simple-array character (*)) trans side)
      (type (f2cl-lib:integer4) info ldc lda k n m)
      (type (array f2cl-lib:complex16 (*)) work c tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
        (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
        (c f2cl-lib:complex16 c-%data% c-%offset%)
        (work f2cl-lib:complex16 work-%data% work-%offset%)
        (side character side-%data% side-%offset%)
        (trans character trans-%data% trans-%offset%))
      (prog
        ((aii #C(0.0d0 0.0d0))
         (taui #C(0.0d0 0.0d0)) (i 0) (i1 0) (i2 0) (i3 0)
         (ic 0) (jc 0) (mi 0) (ni 0) (nq 0) (left nil) (notran nil))
        (declare (type (f2cl-lib:complex16) taui aii)
          (type (f2cl-lib:integer4) nq ni mi jc ic i3 i2 i1 i)
          (type f2cl-lib:logical notran left))
        (setf info 0)
        (setf left
          (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
            (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
        (setf notran
          (multiple-value-bind (ret-val var-0 var-1) (lsame trans "N")
            (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val))
        (cond (left (setf nq m)) (t (setf nq n)))
        (cond
          ((and (not left)
            (not
              (multiple-value-bind (ret-val var-0 var-1) (lsame side "R")

```

```

      (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)))
    (setf info -1))
  ((and (not notran)
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame trans "C")
        (declare (ignore var-1))
        (when var-0 (setf trans var-0)) ret-val)))
    (setf info -2))
  ((< m 0) (setf info -3)) ((< n 0) (setf info -4))
  ((or (< k 0) (> k nq)) (setf info -5))
  ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nq)))
    (setf info -7))
  ((< ldc (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m)))
    (setf info -10)))
  (cond ((/= info 0)
    (xerbla "ZUNM2R" (f2cl-lib:int-sub info)) (go end_label)))
  (if (or (= m 0) (= n 0) (= k 0)) (go end_label))
  (cond
    ((or (and left (not notran)) (and (not left) notran))
      (setf i1 1)
      (setf i2 k)
      (setf i3 1))
    (t (setf i1 k) (setf i2 1) (setf i3 -1)))
  (cond (left (setf ni n) (setf jc 1)) (t (setf mi m) (setf ic 1)))
  (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
    ((> i i2) nil)
    (tagbody
      (cond
        (left (setf mi (f2cl-lib:int-add
          (f2cl-lib:int-sub m i) 1)) (setf ic i))
        (t (setf ni (f2cl-lib:int-add
          (f2cl-lib:int-sub n i) 1)) (setf jc i))))
      (cond
        (notran
          (setf taui
            (f2cl-lib:fref tau-%data% (i) ((1 *)) tau-%offset%)))
        (t
          (setf taui
            (coerce
              (f2cl-lib:dconjg (f2cl-lib:fref tau-%data% (i)
                ((1 *)) tau-%offset%))
              'f2cl-lib:complex16))))
          (setf aii (f2cl-lib:fref a-%data% (i i)
            ((1 lda) (1 *)) a-%offset%))
          (setf (f2cl-lib:fref a-%data% (i i)
            ((1 lda) (1 *)) a-%offset%) one)
          (multiple-value-bind (var-0 var-1 var-2 var-3
            var-4 var-5 var-6 var-7 var-8)
            (zlarf side mi ni
              (f2cl-lib:array-slice a-%data%

```

```

        f2cl-lib:complex16 (i i) ((1 lda) (1 *))
        a-%offset%)
1 tau1
(f2cl-lib:array-slice c-%data%
  f2cl-lib:complex16 (ic jc) ((1 ldc) (1 *))
  c-%offset%)
ldc work)
(declare (ignore var-1 var-2 var-3 var-4 var-5 var-6 var-8))
(setf side var-0) (setf ldc var-7))
(setf (f2cl-lib:fref a-%data% (i i)
  ((1 lda) (1 *)) a-%offset%) aii) label10)
)
(go end_label) end_label
(return (values side trans nil nil nil nil nil nil ldc nil info)))
)))

```

—————→

zunmhr LAPACK

— zunmhr.input —

```

)set break resume
)sys rm -f zunmhr.output
)spool zunmhr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

—————→

— zunmhr.help —

```

=====
zunmhr examples
=====

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:

=====

```

SUBROUTINE ZUNMHR( SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C,
                  LDC, WORK, LWORK, INFO )

.. Scalar Arguments ..
CHARACTER          SIDE, TRANS
INTEGER            IHI, ILO, INFO, LDA, LDC, LWORK, M, N
..
.. Array Arguments ..
COMPLEX*16         A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
..

```

Purpose:

=====

ZUNMHR overwrites the general complex M-by-N matrix C with

| | | |
|--------------|--------------|--------------|
| | SIDE = 'L' | SIDE = 'R' |
| TRANS = 'N': | $Q * C$ | $C * Q$ |
| TRANS = 'C': | $Q^{*H} * C$ | $C * Q^{*H}$ |

where Q is a complex unitary matrix of order nq, with $nq = m$ if SIDE = 'L' and $nq = n$ if SIDE = 'R'. Q is defined as the product of IHI-ILO elementary reflectors, as returned by ZGEHRD:

$$Q = H(i\text{lo}) H(i\text{lo}+1) \dots H(i\text{hi}-1).$$

Arguments:

=====

[in] SIDE

SIDE is CHARACTER*1
 = 'L': apply Q or Q^{*H} from the Left;
 = 'R': apply Q or Q^{*H} from the Right.

[in] TRANS

TRANS is CHARACTER*1
 = 'N': apply Q (No transpose)
 = 'C': apply Q^{*H} (Conjugate transpose)

[in] M

M is INTEGER
 The number of rows of the matrix C. $M \geq 0$.

[in] N

N is INTEGER
The number of columns of the matrix C. $N \geq 0$.

[in] ILO

ILO is INTEGER

[in] IHI

IHI is INTEGER

ILO and IHI must have the same values as in the previous call of ZGEHRD. Q is equal to the unit matrix except in the submatrix $Q(\text{ilo}+1:\text{ihi}, \text{ilo}+1:\text{ihi})$.
If SIDE = 'L', then $1 \leq \text{ILO} \leq \text{IHI} \leq M$, if $M > 0$, and $\text{ILO} = 1$ and $\text{IHI} = 0$, if $M = 0$;
if SIDE = 'R', then $1 \leq \text{ILO} \leq \text{IHI} \leq N$, if $N > 0$, and $\text{ILO} = 1$ and $\text{IHI} = 0$, if $N = 0$.

[in] A

A is COMPLEX*16 array, dimension
 (LDA,M) if SIDE = 'L'
 (LDA,N) if SIDE = 'R'
The vectors which define the elementary reflectors, as returned by ZGEHRD.

[in] LDA

LDA is INTEGER
The leading dimension of the array A.
 $\text{LDA} \geq \max(1, M)$ if SIDE = 'L'; $\text{LDA} \geq \max(1, N)$ if SIDE = 'R'.

[in] TAU

TAU is COMPLEX*16 array, dimension
 (M-1) if SIDE = 'L'
 (N-1) if SIDE = 'R'
TAU(i) must contain the scalar factor of the elementary reflector $H(i)$, as returned by ZGEHRD.

[in,out] C

C is COMPLEX*16 array, dimension (LDC,N)
On entry, the M-by-N matrix C.
On exit, C is overwritten by Q^*C or $Q^{**H}C$ or CQ^{**H} or CQ .

[in] LDC

LDC is INTEGER
The leading dimension of the array C. $LDC \geq \max(1, M)$.

[out] WORK

WORK is COMPLEX*16 array, dimension $(\max(1, LWORK))$
On exit, if $INFO = 0$, $WORK(1)$ returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER
The dimension of the array WORK.
If $SIDE = 'L'$, $LWORK \geq \max(1, N)$;
if $SIDE = 'R'$, $LWORK \geq \max(1, M)$.
For optimum performance $LWORK \geq N \cdot NB$ if $SIDE = 'L'$, and
 $LWORK \geq M \cdot NB$ if $SIDE = 'R'$, where NB is the optimal
blocksize.

If $LWORK = -1$, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

[out] INFO

INFO is INTEGER
= 0: successful exit
< 0: if $INFO = -i$, the i-th argument had an illegal value

Authors:
=====

Univ. of Tennessee
Univ. of California Berkeley
Univ. of Colorado Denver
NAG Ltd.

November 2011

— zunmhr.f —

```
* =====
  SUBROUTINE ZUNMHR( SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C,
    $                LDC, WORK, LWORK, INFO )
```

```

*
* -- LAPACK computational routine (version 3.4.0) --
* -- LAPACK is a software package provided by Univ. of Tennessee, --
* -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*   November 2011
*
*   .. Scalar Arguments ..
*   CHARACTER          SIDE, TRANS
*   INTEGER            IHI, ILO, INFO, LDA, LDC, LWORK, M, N
*
*   ..
*   .. Array Arguments ..
*   COMPLEX*16         A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*   ..
*
*   =====
*
*   .. Local Scalars ..
*   LOGICAL            LEFT, LQUERY
*   INTEGER            I1, I2, IINFO, LWKOPT, MI, NB, NH, NI, NQ, NW
*
*   ..
*   .. External Functions ..
*   LOGICAL            LSAME
*   INTEGER            ILAENV
*   EXTERNAL           LSAME, ILAENV
*
*   ..
*   .. External Subroutines ..
*   EXTERNAL           XERBLA, ZUNMQR
*
*   ..
*   .. Intrinsic Functions ..
*   INTRINSIC          MAX, MIN
*
*   ..
*   .. Executable Statements ..
*
*   Test the input arguments
*
*   INFO = 0
*   NH = IHI - ILO
*   LEFT = LSAME( SIDE, 'L' )
*   LQUERY = ( LWORK.EQ.-1 )
*
*   NQ is the order of Q and NW is the minimum dimension of WORK
*
*
*   IF( LEFT ) THEN
*       NQ = M
*       NW = N
*   ELSE
*       NQ = N
*       NW = M
*   END IF
*   IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN

```

```

        INFO = -1
        ELSE IF( .NOT.LSAME( TRANS, 'N' ) .AND. .NOT.LSAME( TRANS, 'C' ) )
$           THEN
            INFO = -2
        ELSE IF( M.LT.0 ) THEN
            INFO = -3
        ELSE IF( N.LT.0 ) THEN
            INFO = -4
        ELSE IF( ILO.LT.1 .OR. ILO.GT.MAX( 1, NQ ) ) THEN
            INFO = -5
        ELSE IF( IHI.LT.MIN( ILO, NQ ) .OR. IHI.GT.NQ ) THEN
            INFO = -6
        ELSE IF( LDA.LT.MAX( 1, NQ ) ) THEN
            INFO = -8
        ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
            INFO = -11
        ELSE IF( LWORK.LT.MAX( 1, NW ) .AND. .NOT.LQUERY ) THEN
            INFO = -13
        END IF
*
        IF( INFO.EQ.0 ) THEN
            IF( LEFT ) THEN
                NB = ILAENV( 1, 'ZUNMQR', SIDE // TRANS, NH, N, NH, -1 )
            ELSE
                NB = ILAENV( 1, 'ZUNMQR', SIDE // TRANS, M, NH, NH, -1 )
            END IF
            LWKOPT = MAX( 1, NW ) * NB
            WORK( 1 ) = LWKOPT
        END IF
*
        IF( INFO.NE.0 ) THEN
            CALL XERBLA( 'ZUNMHR', -INFO )
            RETURN
        ELSE IF( LQUERY ) THEN
            RETURN
        END IF
*
* Quick return if possible
*
        IF( M.EQ.0 .OR. N.EQ.0 .OR. NH.EQ.0 ) THEN
            WORK( 1 ) = 1
            RETURN
        END IF
*
        IF( LEFT ) THEN
            MI = NH
            NI = N
            I1 = ILO + 1
            I2 = 1
        ELSE

```

```

        MI = M
        NI = NH
        I1 = 1
        I2 = ILO + 1
    END IF
*
    CALL ZUNMQR( SIDE, TRANS, MI, NI, NH, A( ILO+1, ILO ), LDA,
$           TAU( ILO ), C( I1, I2 ), LDC, WORK, LWORK, IINFO )
*
    WORK( 1 ) = LWKOPT
    RETURN
*
*   End of ZUNMHR
*
END

```

— LAPACK zunmhr —

```

(defun zunmhr (side trans m n ilo ihi a lda tau c ldc work lwork info)
  (declare (type (simple-array character (*)) trans side)
    (type (f2cl-lib:integer4) info lwork ldc lda ihi ilo n m)
    (type (array f2cl-lib:complex16 (*)) work c tau a))
  (f2cl-lib:with-multi-array-data
    ((a f2cl-lib:complex16 a-%data% a-%offset%)
     (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
     (c f2cl-lib:complex16 c-%data% c-%offset%)
     (work f2cl-lib:complex16 work-%data% work-%offset%)
     (side character side-%data% side-%offset%)
     (trans character trans-%data% trans-%offset%))
    (prog
      ((i1 0) (i2 0) (iinfo 0) (lwkopt 0) (mi 0)
       (nb 0) (nh 0) (ni 0) (nq 0) (nw 0)
       (left nil) (lquery nil))
      (declare (type (f2cl-lib:integer4) nw nq ni nh nb mi lwkopt iinfo i2 i1)
        (type f2cl-lib:logical lquery left))
      (setf info 0) (setf nh (f2cl-lib:int-sub ihi ilo))
      (setf left
        (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
          (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
      (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical))
      (cond (left (setf nq m) (setf nw n)) (t (setf nq n) (setf nw m)))
      (cond
        ((and (not left)
              (not
                (multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
                  (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)))

```

```

(setf info -1))
((and
  (not
    (multiple-value-bind (ret-val var-0 var-1) (lsame trans "N")
      (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val))
    (not
      (multiple-value-bind (ret-val var-0 var-1) (lsame trans "C")
        (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val)))
  (setf info -2))
((< m 0) (setf info -3)) ((< n 0) (setf info -4))
((or (< ilo 1)
  (> ilo (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nq))))
  (setf info -5))
((or (< ihi (min (the f2cl-lib:integer4 ilo)
  (the f2cl-lib:integer4 nq)))
  (> ihi nq))
  (setf info -6))
((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nq)))
  (setf info -8))
((< ldc (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m)))
  (setf info -11))
((and (< lwork (max (the f2cl-lib:integer4 1)
  (the f2cl-lib:integer4 nw)))
  (not lquery))
  (setf info -13)))
(cond
  ((= info 0)
    (cond
      (left
        (setf nb
          (multiple-value-bind (ret-val var-0 var-1 var-2
            var-3 var-4 var-5 var-6)
            (ilaenv 1 "ZUNMQR" (f2cl-lib:f2cl-// side trans) nh n nh -1)
            (declare (ignore var-0 var-1 var-2 var-6))
            (when var-3 (setf nh var-3))
            (when var-4 (setf n var-4))
            (when var-5 (setf nh var-5)) ret-val))))
        (t
          (setf nb
            (multiple-value-bind (ret-val var-0 var-1 var-2 var-3
              var-4 var-5 var-6)
              (ilaenv 1 "ZUNMQR" (f2cl-lib:f2cl-// side trans) m nh nh -1)
              (declare (ignore var-0 var-1 var-2 var-6))
              (when var-3 (setf m var-3))
              (when var-4 (setf nh var-4))
              (when var-5 (setf nh var-5)) ret-val))))))
    (setf lwkopt
      (f2cl-lib:int-mul
        (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nw)) nb))
    (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)

```

```

      (coerce lwkopt 'f2cl-lib:complex16))))
(cond ((/= info 0)
      (xerbla "ZUNMHR" (f2cl-lib:int-sub info)) (go end_label))
      (lquery (go end_label)))
(cond
  ((or (= m 0) (= n 0) (= nh 0))
   (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
         (coerce 1 'f2cl-lib:complex16))
   (go end_label)))
(cond
  (left (setf mi nh) (setf ni n) (setf i1 (f2cl-lib:int-add ilo 1))
        (setf i2 1))
  (t (setf mi m)
      (setf ni nh)
      (setf i1 1)
      (setf i2 (f2cl-lib:int-add ilo 1))))
(multiple-value-bind
  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7
   var-8 var-9 var-10 var-11
   var-12)
  (zunmqr side trans mi ni nh
   (f2cl-lib:array-slice a-%data% f2cl-lib:complex16 ((+ ilo 1) ilo)
    ((1 lda) (1 *)) a-%offset%)
   lda
   (f2cl-lib:array-slice tau-%data% f2cl-lib:complex16 (ilo) ((1 *))
    tau-%offset%)
   (f2cl-lib:array-slice c-%data%
    f2cl-lib:complex16 (i1 i2) ((1 ldc) (1 *))
    c-%offset%)
   ldc work lwork iinfo)
  (declare (ignore var-5 var-7 var-8 var-10))
  (when var-0 (setf side var-0))
  (when var-1 (setf trans var-1)) (when var-2 (setf mi var-2))
  (when var-3 (setf ni var-3)) (when var-4 (setf nh var-4))
  (when var-6 (setf lda var-6)) (when var-9 (setf ldc var-9))
  (when var-11 (setf lwork var-11)) (when var-12 (setf iinfo var-12)))
  (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
        (coerce lwkopt 'f2cl-lib:complex16))
  (go end_label) end_label
  (return
   (values side trans m n nil nil nil lda nil nil ldc nil lwork info)))
))

```

zunmqr LAPACK

— zunmqr.input —

```

)set break resume
)sys rm -f zunmqr.output
)spool zunmqr.output
)set message test on
)set message auto off
)clear all

)spool
)lisp (bye)

```

— zunmqr.help —

```

=====
zunmqr examples
=====

```

```

=====
Man Page Details
=====

```

Online html documentation available at
<http://www.netlib.org/lapack/explore-html/>

Definition:
 =====

```

SUBROUTINE ZUNMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
                  WORK, LWORK, INFO )

  .. Scalar Arguments ..
  CHARACTER          SIDE, TRANS
  INTEGER            INFO, K, LDA, LDC, LWORK, M, N
  ..
  .. Array Arguments ..
  COMPLEX*16         A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
  ..

```

Purpose:
 =====

ZUNMQR overwrites the general complex M-by-N matrix C with

```

                                SIDE = 'L'      SIDE = 'R'
TRANS = 'N':      Q * C          C * Q

```

TRANS = 'C': Q**H * C C * Q**H

where Q is a complex unitary matrix defined as the product of k elementary reflectors

$$Q = H(1) H(2) \dots H(k)$$

as returned by ZGEQRF. Q is of order M if SIDE = 'L' and of order N if SIDE = 'R'.

Arguments:

=====

[in] SIDE

SIDE is CHARACTER*1
 = 'L': apply Q or Q**H from the Left;
 = 'R': apply Q or Q**H from the Right.

[in] TRANS

TRANS is CHARACTER*1
 = 'N': No transpose, apply Q;
 = 'C': Conjugate transpose, apply Q**H.

[in] M

M is INTEGER
 The number of rows of the matrix C. M >= 0.

[in] N

N is INTEGER
 The number of columns of the matrix C. N >= 0.

[in] K

K is INTEGER
 The number of elementary reflectors whose product defines the matrix Q.
 If SIDE = 'L', M >= K >= 0;
 if SIDE = 'R', N >= K >= 0.

[in] A

A is COMPLEX*16 array, dimension (LDA,K)
 The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by ZGEQRF in the first k columns of its array argument A.
 A is modified by the routine but restored on exit.

[in] LDA

LDA is INTEGER
The leading dimension of the array A.
If SIDE = 'L', LDA \geq max(1,M);
if SIDE = 'R', LDA \geq max(1,N).

[in] TAU

TAU is COMPLEX*16 array, dimension (K)
TAU(i) must contain the scalar factor of the elementary
reflector H(i), as returned by ZGEQRF.

[in,out] C

C is COMPLEX*16 array, dimension (LDC,N)
On entry, the M-by-N matrix C.
On exit, C is overwritten by Q*C or Q**H*C or C*Q**H or C*Q.

[in] LDC

LDC is INTEGER
The leading dimension of the array C. LDC \geq max(1,M).

[out] WORK

WORK is COMPLEX*16 array, dimension (MAX(1,LWORK))
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

[in] LWORK

LWORK is INTEGER
The dimension of the array WORK.
If SIDE = 'L', LWORK \geq max(1,N);
if SIDE = 'R', LWORK \geq max(1,M).
For optimum performance LWORK \geq N*NB if SIDE = 'L', and
LWORK \geq M*NB if SIDE = 'R', where NB is the optimal
blocksize.

If LWORK = -1, then a workspace query is assumed; the routine
only calculates the optimal size of the WORK array, returns
this value as the first entry of the WORK array, and no error
message related to LWORK is issued by XERBLA.

[out] INFO

INFO is INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value

Authors:

=====

Univ. of Tennessee
 Univ. of California Berkeley
 Univ. of Colorado Denver
 NAG Ltd.

November 2011

— zunmqr.f —

```
* =====
*      SUBROUTINE ZUNMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
*      $                  WORK, LWORK, INFO )
*
*      -- LAPACK computational routine (version 3.4.0) --
*      -- LAPACK is a software package provided by Univ. of Tennessee, --
*      -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd.--
*      November 2011
*
*      .. Scalar Arguments ..
*      CHARACTER          SIDE, TRANS
*      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
*
*      ..
*      .. Array Arguments ..
*      COMPLEX*16         A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )
*
*      ..
*
*      =====
*
*      .. Parameters ..
*      INTEGER            NBMAX, LDT
*      PARAMETER          ( NBMAX = 64, LDT = NBMAX+1 )
*
*      ..
*      .. Local Scalars ..
*      LOGICAL            LEFT, LQUERY, NOTRAN
*      INTEGER            I, I1, I2, I3, IB, IC, IINFO, IWS, JC, LDWORK,
*      $                  LWKOPT, MI, NB, NBMIN, NI, NQ, NW
*
*      ..
*      .. Local Arrays ..
*      COMPLEX*16         T( LDT, NBMAX )
*
*      ..
*      .. External Functions ..
*      LOGICAL            LSAME
```

```

      INTEGER          ILAENV
      EXTERNAL         LSAME, ILAENV
*
*   ..
*   .. External Subroutines ..
      EXTERNAL         XERBLA, ZLARFB, ZLARFT, ZUNM2R
*
*   ..
*   .. Intrinsic Functions ..
      INTRINSIC         MAX, MIN
*
*   ..
*   .. Executable Statements ..
*
      Test the input arguments
*
      INFO = 0
      LEFT = LSAME( SIDE, 'L' )
      NOTRAN = LSAME( TRANS, 'N' )
      LQUERY = ( LWORK.EQ.-1 )
*
*   NQ is the order of Q and NW is the minimum dimension of WORK
*
      IF( LEFT ) THEN
        NQ = M
        NW = N
      ELSE
        NQ = N
        NW = M
      END IF
      IF( .NOT.LEFT .AND. .NOT.LSAME( SIDE, 'R' ) ) THEN
        INFO = -1
      ELSE IF( .NOT.NOTRAN .AND. .NOT.LSAME( TRANS, 'C' ) ) THEN
        INFO = -2
      ELSE IF( M.LT.0 ) THEN
        INFO = -3
      ELSE IF( N.LT.0 ) THEN
        INFO = -4
      ELSE IF( K.LT.0 .OR. K.GT.NQ ) THEN
        INFO = -5
      ELSE IF( LDA.LT.MAX( 1, NQ ) ) THEN
        INFO = -7
      ELSE IF( LDC.LT.MAX( 1, M ) ) THEN
        INFO = -10
      ELSE IF( LWORK.LT.MAX( 1, NW ) .AND. .NOT.LQUERY ) THEN
        INFO = -12
      END IF
*
      IF( INFO.EQ.0 ) THEN
*
*       Determine the block size.  NB may be at most NBMAX, where NBMAX
*       is used to define the local array T.
*

```

```

      NB = MIN( NBMAX, ILAENV( 1, 'ZUNMQR', SIDE // TRANS, M, N, K,
$      -1 ) )
      LWKOPT = MAX( 1, NW )*NB
      WORK( 1 ) = LWKOPT
      END IF
*
      IF( INFO.NE.0 ) THEN
        CALL XERBLA( 'ZUNMQR', -INFO )
        RETURN
      ELSE IF( LQUERY ) THEN
        RETURN
      END IF
*
*      Quick return if possible
*
      IF( M.EQ.0 .OR. N.EQ.0 .OR. K.EQ.0 ) THEN
        WORK( 1 ) = 1
        RETURN
      END IF
*
      NBMIN = 2
      LDWORK = NW
      IF( NB.GT.1 .AND. NB.LT.K ) THEN
        IWS = NW*NB
        IF( LWORK.LT.IWS ) THEN
          NB = LWORK / LDWORK
          NBMIN = MAX( 2, ILAENV( 2, 'ZUNMQR', SIDE // TRANS, M, N, K,
$          -1 ) )
        END IF
      ELSE
        IWS = NW
      END IF
*
      IF( NB.LT.NBMIN .OR. NB.GE.K ) THEN
*
*      Use unblocked code
*
        CALL ZUNM2R( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
$          IINFO )
      ELSE
*
*      Use blocked code
*
        IF( ( LEFT .AND. .NOT.NOTRAN ) .OR.
$        ( .NOT.LEFT .AND. NOTRAN ) ) THEN
          I1 = 1
          I2 = K
          I3 = NB
        ELSE
          I1 = ( ( K-1 ) / NB ) * NB + 1

```

```

        I2 = 1
        I3 = -NB
    END IF
*
    IF( LEFT ) THEN
        NI = N
        JC = 1
    ELSE
        MI = M
        IC = 1
    END IF
*
    DO 10 I = I1, I2, I3
        IB = MIN( NB, K-I+1 )
*
*       Form the triangular factor of the block reflector
*       H = H(i) H(i+1) . . . H(i+ib-1)
*
        CALL ZLARFT( 'Forward', 'Columnwise', NQ-I+1, IB, A( I, I ),
$           LDA, TAU( I ), T, LDT )
        IF( LEFT ) THEN
*
*           H or H**H is applied to C(i:m,1:n)
*
*           MI = M - I + 1
*           IC = I
        ELSE
*
*           H or H**H is applied to C(1:m,i:n)
*
*           NI = N - I + 1
*           JC = I
        END IF
*
        Apply H or H**H
*
        CALL ZLARFB( SIDE, TRANS, 'Forward', 'Columnwise', MI, NI,
$           IB, A( I, I ), LDA, T, LDT, C( IC, JC ), LDC,
$           WORK, LDWORK )
10    CONTINUE
    END IF
    WORK( 1 ) = LWKOPT
    RETURN
*
*   End of ZUNMQR
*
END

```

```

Warning: Types of argument 0 in call to ZUNM2R do not match.
Declared type: (SIMPLE-ARRAY CHARACTER (*))
Argument type: (STRING 1)
Warning: Types of argument 1 in call to ZUNM2R do not match.
Declared type: (SIMPLE-ARRAY CHARACTER (*))
Argument type: (STRING 1)
Warning: Types of argument 0 in call to ZLARFB do not match.
Declared type: (SIMPLE-ARRAY CHARACTER (*))
Argument type: (STRING 1)
Warning: Types of argument 1 in call to ZLARFB do not match.
Declared type: (SIMPLE-ARRAY CHARACTER (*))
Argument type: (STRING 1)

```

— LAPACK zunmqr —

```

(let* ((nbmax 64) (ldt (+ nbmax 1)))
  (declare (type (f2cl-lib:integer4 64 64) nbmax) (type (f2cl-lib:integer4) ldt)
    (ignorable nbmax ldt))
  (defun zunmqr (side trans m n k a lda tau c ldc work lwork info)
    (declare (type (simple-array character (*)) trans side)
      (type (f2cl-lib:integer4) info lwork ldc lda k n m)
      (type (array f2cl-lib:complex16 (*)) work c tau a))
    (f2cl-lib:with-multi-array-data
      ((a f2cl-lib:complex16 a-%data% a-%offset%)
       (tau f2cl-lib:complex16 tau-%data% tau-%offset%)
       (c f2cl-lib:complex16 c-%data% c-%offset%)
       (work f2cl-lib:complex16 work-%data% work-%offset%)
       (side character side-%data% side-%offset%)
       (trans character trans-%data% trans-%offset%))
      (prog
        ((i 0) (i1 0) (i2 0) (i3 0) (ib 0) (ic 0)
         (iinfo 0) (iws 0) (jc 0) (ldwork 0)
         (lwkopt 0) (mi 0) (nb 0) (nbmin 0) (ni 0) (nq 0) (nw 0) (left nil)
         (lquery nil) (notran nil)
         (t$
          (make-array (the fixnum (reduce #'* (list ldt nbmax))) :element-type
            'f2cl-lib:complex16)))
        (declare
          (type (f2cl-lib:integer4) nw nq ni nbmin nb mi
            lwkopt ldwork jc iws iinfo ic
            ib i3 i2 i1 i)
          (type f2cl-lib:logical notran lquery left)
          (type (array f2cl-lib:complex16 (*)) t$))
        (setf info 0)
        (setf left
          (multiple-value-bind (ret-val var-0 var-1) (lsame side "L")
            (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val))
        (setf notran
          (multiple-value-bind (ret-val var-0 var-1) (lsame trans "N")

```

```

      (declare (ignore var-1)) (when var-0 (setf trans var-0)) ret-val))
    (setf lquery (coerce (= lwork -1) 'f2cl-lib:logical))
    (cond (left (setf nq m) (setf nw n)) (t (setf nq n) (setf nw m)))
    (cond
      ((and (not left)
            (not
              (multiple-value-bind (ret-val var-0 var-1) (lsame side "R")
                (declare (ignore var-1)) (when var-0 (setf side var-0)) ret-val)))
        (setf info -1))
      ((and (not notran)
            (not
              (multiple-value-bind (ret-val var-0 var-1) (lsame trans "C")
                (declare (ignore var-1))
                (when var-0 (setf trans var-0)) ret-val)))
        (setf info -2))
      ((< m 0) (setf info -3)) ((< n 0) (setf info -4))
      ((or (< k 0) (> k nq)) (setf info -5))
      ((< lda (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nq)))
        (setf info -7))
      ((< ldc (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 m)))
        (setf info -10))
      ((and (< lwork (max (the f2cl-lib:integer4 1)
                          (the f2cl-lib:integer4 nw)))
            (not lquery))
        (setf info -12)))
    (cond
      ((= info 0)
        (setf nb
          (min (the f2cl-lib:integer4 nbmax)
              (the f2cl-lib:integer4
                (multiple-value-bind (ret-val var-0 var-1 var-2
                                      var-3 var-4 var-5 var-6)
                  (ilaenv 1 "ZUNMQR" (f2cl-lib:f2cl-// side trans) m n k -1)
                  (declare (ignore var-0 var-1 var-2 var-6))
                  (when var-3 (setf m var-3))
                  (when var-4 (setf n var-4))
                  (when var-5 (setf k var-5)) ret-val))))))
        (setf lwkopt
          (f2cl-lib:int-mul
            (max (the f2cl-lib:integer4 1) (the f2cl-lib:integer4 nw)) nb))
        (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
          (coerce lwkopt 'f2cl-lib:complex16))))
      (cond ((/= info 0)
              (xerbla "ZUNMQR" (f2cl-lib:int-sub info)) (go end_label))
            (lquery (go end_label)))
      (cond
        ((or (= m 0) (= n 0) (= k 0))
          (setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
            (coerce 1 'f2cl-lib:complex16))
          (go end_label)))

```

```

(setf nbmin 2) (setf ldwork nw)
(cond
  ((and (> nb 1) (< nb k)) (setf iws (f2cl-lib:int-mul nw nb))
    (cond
      ((< lwork iws)
        (setf nb (the f2cl-lib:integer4 (truncate lwork ldwork)))
        (setf nbmin
          (max (the f2cl-lib:integer4 2)
            (the f2cl-lib:integer4
              (multiple-value-bind
                (ret-val var-0 var-1 var-2 var-3 var-4 var-5 var-6)
                (ilaenv 2 "ZUNMQR" (f2cl-lib:f2cl-// side trans) m n k -1)
                (declare (ignore var-0 var-1 var-2 var-6))
                (when var-3 (setf m var-3))
                (when var-4 (setf n var-4))
                (when var-5 (setf k var-5)) ret-val)))))))
      (t (setf iws nw)))
    (cond
      ((or (< nb nbmin) (>= nb k))
        (multiple-value-bind
          (var-0 var-1 var-2 var-3 var-4 var-5
           var-6 var-7 var-8 var-9 var-10 var-11)
          (zunm2r side trans m n k a lda tau c ldc work iinfo)
          (declare (ignore var-2 var-3 var-4 var-5 var-6 var-7 var-8 var-10))
          (setf side var-0)
          (setf trans var-1)
          (setf ldc var-9)
          (setf iinfo var-11)))
        (t
          (cond
            ((or (and left (not notran)) (and (not left) notran)) (setf i1 1)
              (setf i2 k) (setf i3 nb))
            (t (setf i1
              (+ (* (the f2cl-lib:integer4 (truncate (- k 1) nb)) nb) 1))
              (setf i2 1) (setf i3 (f2cl-lib:int-sub nb))))
            (cond (left (setf ni n) (setf jc 1)) (t (setf mi m) (setf ic 1)))
            (f2cl-lib:fdo (i i1 (f2cl-lib:int-add i i3))
              (> i i2) nil)
              (tagbody
                (setf ib
                  (min (the f2cl-lib:integer4 nb)
                    (the f2cl-lib:integer4 (f2cl-lib:int-add
                      (f2cl-lib:int-sub k i) 1))))
                (multiple-value-bind
                  (var-0 var-1 var-2 var-3 var-4 var-5 var-6 var-7 var-8)
                  (zlarft "Forward" "Columnwise"
                    (f2cl-lib:int-add (f2cl-lib:int-sub nq i) 1) ib
                    (f2cl-lib:array-slice a-%data%
                      f2cl-lib:complex16 (i i) ((1 lda) (1 *))
                      a-%offset%))

```



```

lda
(f2cl-lib:array-slice tau-%data%
  f2cl-lib:complex16 (i) ((1 *))
  tau-%offset%)
t$ ldt)
(declare (ignore var-0 var-1 var-2 var-3
  var-4 var-6 var-7))
(setf lda var-5) (setf ldt var-8))
(cond
(left (setf mi (f2cl-lib:int-add
  (f2cl-lib:int-sub m i) 1)) (setf ic i))
(t (setf ni (f2cl-lib:int-add
  (f2cl-lib:int-sub n i) 1)) (setf jc i)))
(multiple-value-bind
(var-0 var-1 var-2 var-3 var-4 var-5
  var-6 var-7 var-8 var-9 var-10 var-11
  var-12 var-13 var-14)
(zlarfb side trans "Forward" "Columnwise" mi ni ib
  (f2cl-lib:array-slice a-%data%
    f2cl-lib:complex16 (i i) ((1 lda) (1 *))
    a-%offset%)
  lda t$ ldt
  (f2cl-lib:array-slice c-%data%
    f2cl-lib:complex16 (ic jc) ((1 ldc) (1 *))
    c-%offset%)
  ldc work ldwork)
(declare (ignore var-2 var-3 var-4 var-5
  var-7 var-9 var-11 var-13))
(setf side var-0)
(setf trans var-1)
(setf ib var-6)
(setf lda var-8)
(setf ldt var-10)
(setf ldc var-12)
(setf ldwork var-14))
label10)))
(setf (f2cl-lib:fref work-%data% (1) ((1 *)) work-%offset%)
  (coerce lwkopt 'f2cl-lib:complex16))
(go end_label) end_label
(return
  (values side trans m n k nil lda nil nil ldc nil nil info))))

```

Chapter 7

LAPACK tests

```
;;;
;;; Simple tests for selected LAPACK routines.
;;;
;;; $Id$
;;;

(in-package "LAPACK")

;; Convert the eigenvalues returned by DGEEV into an array
(defun make-eigval (wr wi)
  (let ((e-val (make-array (length wr))))
    (map-into e-val #'(lambda (r i)
      ;; Do we really want to do this? Should we
      ;; just make all of the eigenvalues complex?
      (if (zerop i)
          r
          (complex r i)))
      wr wi)
    e-val))

;; Convert the eigenvalues returned by DGEEV into a more typical
;; matrix form.
(defun make-eigvec (n vr wi)
  (let ((evec (make-array (list n n))))
    (do ((col 0 (incf col))
        (posn 0))
      ((>= col n))
      (cond ((zerop (aref wi col))
             (dotimes (row n)
               (setf (aref evec row col) (aref vr posn))
               (incf posn)))
            (t
```

```

      (dotimes (row n)
        (let* ((next-posn (+ posn n))
              (val+ (complex (aref vr posn) (aref vr next-posn)))
              (val- (conjugate val+)))
          (setf (aref evec row col) val+)
          (setf (aref evec row (1+ col)) val-)
          (incf posn)))
        ;; Skip over the next column, which we've already used
        (incf col)
        (incf posn n))))
      evec))

;; Expected results from
;; http://www.nag.co.uk/lapack-ex/examples/results/dgeev-ex.r
;;
;; DGEEV Example Program Results
;;
;; Eigenvalue( 1) = 7.9948E-01
;;
;; Eigenvector( 1)
;; -6.5509E-01
;; -5.2363E-01
;; 5.3622E-01
;; -9.5607E-02
;;
;; Eigenvalue( 2) = (-9.9412E-02, 4.0079E-01)
;;
;; Eigenvector( 2)
;; (-1.9330E-01, 2.5463E-01)
;; ( 2.5186E-01, -5.2240E-01)
;; ( 9.7182E-02, -3.0838E-01)
;; ( 6.7595E-01, 0.0000E+00)
;;
;; Eigenvalue( 3) = (-9.9412E-02, -4.0079E-01)
;;
;; Eigenvector( 3)
;; (-1.9330E-01, -2.5463E-01)
;; ( 2.5186E-01, 5.2240E-01)
;; ( 9.7182E-02, 3.0838E-01)
;; ( 6.7595E-01, -0.0000E+00)
;;
;; Eigenvalue( 4) = -1.0066E-01
;;
;; Eigenvector( 4)
;; 1.2533E-01
;; 3.3202E-01
;; 5.9384E-01
;; 7.2209E-01
;;
(defun print-dgeev-results (e-val e-vec)

```

```

(format t "~2%DGEEV Example Program Results~%")
(let ((n (length e-val)))
  (dotimes (k n)
    (format t "Eigenvalue(~D) = ~A~%" k (aref e-val k))
    (format t "~%Eigenvector(~D)~%" k)
    (dotimes (row n)
      (format t "~A~%" (aref e-vec row k))
      (terpri))))

(defun check-eigen-val-vec (n e-val e-vec true-val true-vec &key (tol 1d-14))
  (flet ((relerr-ok (est true)
    (let* ((re (/ (abs (- est true))
      (abs true)))
      (ok (<= re tol)))
      ;; Return NIL if it's ok. Otherwise return a list to
      ;; indicate what failed.
      (unless ok
        (format t "est = ~S~%true = ~S~% rel = ~S~%"
          est true re)
        (list est true re))))))
    (or (relerr-ok (aref e-val n) true-val)
      (dotimes (k n t)
        (let ((res (relerr-ok (aref e-vec k n) (aref true-vec k))))
          (when res
            (return res)))))))

;; DGEEV example based on the example from
;; http://www.nag.co.uk/lapack-ex/node87.html
(defun test-dgeev ()
  ;; The matrix is
  ;;
  ;; 0.35 0.45 -0.14 -0.17
  ;; 0.09 0.07 -0.54 0.35
  ;; -0.44 -0.33 -0.03 0.17
  ;; 0.25 -0.32 -0.13 0.11
  ;;
  ;; Recall that Fortran arrays are column-major order!
  (let* ((n 4)
    (a-mat (make-array (* n n) :element-type 'double-float
      :initial-contents '(0.35d0 0.09d0 -0.44d0 0.25d0
        0.45d0 0.07d0 -0.33d0 -0.32d0
        -0.14d0 -0.54d0 -0.03d0 -0.13d0
        -0.17d0 0.35d0 0.17d0 0.11d0)))
    (wr (make-array n :element-type 'double-float))
    (wi (make-array n :element-type 'double-float))
    (vl (make-array 0 :element-type 'double-float))
    (vr (make-array (* n n) :element-type 'double-float))
    (lwork 660)
    (work (make-array lwork :element-type 'double-float)))
    (multiple-value-bind (z-jobvl z-jobvr z-n z-a z-lda z-wr z-wi z-vl z-ldvl

```

```

                                z-vr z-ldvr z-work z-lwork info)
(dgeev "N" "V" n a-mat n wr wi vl n vr n work lwork 0)
  (declare (ignore z-jobvl z-jobvr z-n z-a z-lda z-wr z-wi z-vl z-ldvl z-vr
    z-ldvr z-work z-lwork))
  (let ((e-val (make-eigval wr wi))
    (e-vec (make-eigvec n vr wi)))
    ;; Display solution
    (cond ((zerop info)
      (print-dgeev-results e-val
        e-vec))
      (t
        (format t "Failure in DGEEV. INFO = ~D~%" info)))
    ;; Display workspace info
    (format t "Optimum workspace required = ~D~%" (truncate (aref work 0)))
    (format t "Workspace provided = ~D~%" lwork)

    (values e-val e-vec))))))

(rt:deftest dgeev.1
  (multiple-value-bind (e-val e-vec)
    (test-dgeev)
    (list (check-eigen-val-vec 0 e-val e-vec
      0.799482122586210d0
      #(-0.6550887675124076d0
        -0.5236294609021240d0
        0.5362184613722345d0
        -0.0956067782012298d0))
      (check-eigen-val-vec 1 e-val e-vec
        #c(-0.0994124532950747d0 0.4007924719897546d0)
        #(#c(-0.193301548264222d0 0.254631571927584d0)
          #c(0.251856531726740d0 -0.522404734711629d0)
          #c(0.097182458443282d0 -0.308383755897228d0)
          #c(0.675954054254748 0d0))))
      (check-eigen-val-vec 2 e-val e-vec
        #c(-0.0994124532950747d0 -0.4007924719897546d0)
        #(#c(-0.193301548264222d0 -0.254631571927584d0)
          #c(0.251856531726740d0 0.522404734711629d0)
          #c(0.097182458443282d0 0.308383755897228d0)
          #c(0.675954054254748 0d0))))
      (check-eigen-val-vec 3 e-val e-vec
        -0.100657215996059d0
        #(0.125332697230903d0
          0.332022215571751d0
          0.593837759557331d0
          0.722087029862455d0
          -0.6550887675124076d0))))
    (t t t t)))

;; Expected results http://www.nag.co.uk/lapack-ex/examples/results/dgeevx-ex.r
;;

```

```

;; DGEEVX Example Program Results
;;
;; Eigenvalue( 1) = 7.9948E-01
;;
;; Reciprocal condition number = 9.9E-01
;; Error bound = 1.3E-16
;;
;; Eigenvector( 1)
;; -6.5509E-01
;; -5.2363E-01
;; 5.3622E-01
;; -9.5607E-02
;;
;; Reciprocal condition number = 8.2E-01
;; Error bound = 1.6E-16
;;
;; Eigenvalue( 2) = (-9.9412E-02, 4.0079E-01)
;;
;; Reciprocal condition number = 7.0E-01
;; Error bound = 1.8E-16
;;
;; Eigenvector( 2)
;; (-1.9330E-01, 2.5463E-01)
;; ( 2.5186E-01,-5.2240E-01)
;; ( 9.7182E-02,-3.0838E-01)
;; ( 6.7595E-01, 0.0000E+00)
;;
;; Reciprocal condition number = 4.0E-01
;; Error bound = 3.3E-16
;;
;; Eigenvalue( 3) = (-9.9412E-02,-4.0079E-01)
;;
;; Reciprocal condition number = 7.0E-01
;; Error bound = 1.8E-16
;;
;; Eigenvector( 3)
;; (-1.9330E-01,-2.5463E-01)
;; ( 2.5186E-01, 5.2240E-01)
;; ( 9.7182E-02, 3.0838E-01)
;; ( 6.7595E-01,-0.0000E+00)
;;
;; Reciprocal condition number = 4.0E-01
;; Error bound = 3.3E-16
;;
;; Eigenvalue( 4) = -1.0066E-01
;;
;; Reciprocal condition number = 5.7E-01
;; Error bound = 2.3E-16
;;
;; Eigenvector( 4)

```

```

;; 1.2533E-01
;; 3.3202E-01
;; 5.9384E-01
;; 7.2209E-01
;;
;; Reciprocal condition number = 3.1E-01
;; Error bound = 4.2E-16
;;
(defun print-dgeevx-results (tol e-val e-vec rconde rcondv)
  (format t "~2%DGEEVX Example Program Results~%")
  (let ((n (length e-val)))
    (dotimes (k n)
      (format t "Eigenvalue(~D) = ~A~%" k (aref e-val k))
      (let ((rcnd (aref rconde k)))
        (format t "Reciprocal condition number = ~A~%" rcnd)
        (if (plusp rcnd)
            (format t "Error bound = ~A~%" (/ tol rcnd))
            (format t "Error bound is infinite~%"))))

      (format t "~%Eigenvector(~D)~%" k)
      (dotimes (row n)
        (format t "~A~%" (aref e-vec row k)))
        (let ((rcnd (aref rcondv k)))
          (format t "Reciprocal condition number = ~A~%" rcnd)
          (if (plusp rcnd)
              (format t "Error bound = ~A~%" (/ tol rcnd))
              (format t "Error bound is infinity~%"))))
          (terpri))))

(defun test-dgeevx ()
  (let* ((n 4)
        (a-mat (make-array (* n n) :element-type 'double-float
                           :initial-contents '(0.35d0 0.09d0 -0.44d0 0.25d0
                                                0.45d0 0.07d0 -0.33d0 -0.32d0
                                                -0.14d0 -0.54d0 -0.03d0 -0.13d0
                                                -0.17d0 0.35d0 0.17d0 0.11d0)))
        (wr (make-array n :element-type 'double-float))
        (wi (make-array n :element-type 'double-float))
        (vl (make-array (* n n) :element-type 'double-float))
        (vr (make-array (* n n) :element-type 'double-float))
        (scale (make-array n :element-type 'double-float))
        (rconde (make-array n :element-type 'double-float))
        (rcondv (make-array n :element-type 'double-float))
        (lwork 660)
        (work (make-array lwork :element-type 'double-float))
        (iwork (make-array (- (* n 2) 2) :element-type 'f2cl-lib::integer4)))
    (multiple-value-bind (z-balanc z-jobvl z-jobvr z-sense z-n z-a z-lda z-wr
                        z-wi z-vl z-ldvl z-vr z-ldvr ilo ihi z-scale abnrm
                        z-rconde z-rcondv z-work z-lwork z-iwork info)
      (dgeevx "Balance" "Vectors (left)" "Vectors (right)"

```



```

"Both reciprocal condition numbers"
n a-mat n wr wi vl n vr n 0 0 scale 0d0 rconde rcondv
work lwork iwork 0)
    (declare (ignore z-balanc z-jobv1 z-jobvr z-sense z-n z-a z-lda z-wr
                    z-wi z-v1 z-ldv1 z-vr z-ldvr z-scale z-rconde z-rcondv
                    z-work z-lwork z-iwork))
    ;; Display solution
    (cond ((zerop info)
            (let* ((eps (dlamch "Eps"))
                   (tol (* eps abnrm)))
              (print-dgeevx-results tol
                                     (make-eigval wr wi)
                                     (make-eigvec n vr wi)
                                     rconde rcondv)))
           (t
            (format t "Failure in DGEEV. INFO = ~D~%" info)))
    ;; Display workspace info
    (format t "Optimum workspace required = ~D~%" (truncate (aref work 0)))
    (format t "Workspace provided = ~D~%" lwork)))

;; Expected results (from
;;   http://www.nag.co.uk/lapack-ex/examples/results/dgesv-ex.r)
;; Solution
;;      1.0000   -1.0000    3.0000   -5.0000
;;
;; Details of factorization
;;      1         2         3         4
;; 1    5.2500   -2.9500   -0.9500   -3.8000
;; 2    0.3429    3.8914    2.3757    0.4129
;; 3    0.3010   -0.4631   -1.5139    0.2948
;; 4   -0.2114   -0.3299    0.0047    0.1314
;;
;; Pivot indices
;;      2         2         3         4
;;
(defun print-dgesv-results (n a b ipiv)
  (format t "~2%DGESV Example Program Results~%" )
  (format t "Solution~%" )
  (dotimes (k n)
    (format t "~21,14e " (aref b k)))
  (format t "~&Details of factorization~%" )
  (dotimes (r n)
    (dotimes (c n)
      (format t "~21,14e" (aref a (+ r (* c n))))))
  (terpri))
  (format t "Pivot indices~%" )
  (dotimes (k n)
    (format t " ~d" (aref ipiv k)))
  (terpri))

```

```

(defun test-dgesv ()
  ;;
  ;; Matrix A:
  ;; 1.80  2.88  2.05 -0.89
  ;; 5.25 -2.95 -0.95 -3.80
  ;; 1.58 -2.69 -2.90 -1.04
  ;; -1.11 -0.66 -0.59  0.80
  ;;
  ;; RHS:
  ;; 9.52 24.35  0.77 -6.22
  (let* ((n 4)
    (a-mat (make-array (* n n) :element-type 'double-float
      :initial-contents '(1.80d0 5.25d0 1.58d0 -1.11d0
2.88d0 -2.95d0 -2.69d0 -0.66d0
2.05d0 -0.95d0 -2.90d0 -0.59d0
-0.89d0 -3.80d0 -1.04d0 0.80d0)))
    (b (make-array n :element-type 'double-float
      :initial-contents '(9.52d0 24.35d0 0.77d0 -6.22d0)))
    (ipiv (make-array n :element-type 'f2cl-lib:integer4)))
    (multiple-value-bind (z-n z-nrhs z-a z-lda z-ipiv z-b z-ldb info)
      (dgesv n 1 a-mat n ipiv b n 0))
      (declare (ignore z-n z-nrhs z-a z-lda z-ipiv z-b z-ldb))
      ;; Display solution
      (cond ((zerop info)
        (print-dgesv-results n a-mat b ipiv))
        (t
          (format t "The (~D, ~D) element of the factor U is zero~%"
            info info))))))

;; Expected results (from )
;;
;; It seems, however, that the result from that page are wrong. At
;; least they seem wrong when I run the actual test program. The main
;; difference is that the singular vectors have the signs of some
;; entries wrong.
;;
;; The result below is what the test program actually produces.

;; DGESDD Example Program Results
;;
;; Singular values
;; 9.9966 3.6831 1.3569 0.5000
;; Left singular vectors
;; 1 2 3 4
;; 1 -0.1921 0.8030 -0.0041 0.5642
;; 2 0.8794 0.3926 0.0752 -0.2587
;; 3 -0.2140 0.2980 -0.7827 -0.5027
;; 4 0.3795 -0.3351 -0.6178 0.6017
;;
;; Right singular vectors by row (first m rows of V**T)

```

```

;;          1          2          3          4          5          6
;;  1  -0.2774 -0.2020 -0.2918  0.0938  0.4213 -0.7816
;;  2   0.6003  0.0301 -0.3348  0.3699 -0.5266 -0.3353
;;  3   0.1277 -0.2805 -0.6453 -0.6781 -0.0413  0.1645
;;  4  -0.1323 -0.7034 -0.1906  0.5399  0.0575  0.3957
;;
;; Error estimate for the singular values
;;      1.1E-15
;;
;; Error estimates for the left singular vectors
;;      1.8E-16      4.8E-16      1.3E-15      1.3E-15
;;
;; Error estimates for the right singular vectors
;;      1.8E-16      4.8E-16      1.3E-15      2.2E-15
;;
(defun print-dgesdd-results (m n s u a)
  (format t "~2%DGESDD Example Program Results~%" )
  (format t "Singular values~%" )
  (dotimes (k m)
    (format t "~20,14e" (aref s k)))
  (format t "~2%Left singular vectors~%" )
  (dotimes (r m)
    (dotimes (c m)
      (format t "~16,7e" (aref u (+ r (* c m))))))
    (terpri))
  (format t "~%Right singular vectors (first m rows of V**T)~%" )
  (dotimes (r m)
    (dotimes (c n)
      (format t "~16,7e" (aref a (+ r (* c m))))))
    (terpri))
  ;; Compute error estimates for the singular vectors
  (let ((serrbd (* (aref s 0) (dlamch "Eps"))))
    (rcondv (make-array m :element-type 'double-float))
    (rcondv (make-array m :element-type 'double-float))
    (uerrbd (make-array m :element-type 'double-float))
    (verrbd (make-array m :element-type 'double-float)))
    (ddisna "Left" m n s rcondv 0)
    (ddisna "Right" m n s rcondv 0)
    (dotimes (k m)
      (setf (aref uerrbd k) (/ serrbd (aref rcondv k)))
      (setf (aref verrbd k) (/ serrbd (aref rcondv k))))
    (format t "Error estimate for the singular values~%" )
    (format t "~20,15g~%" serrbd)
    (format t "~%Error estimates for the left singular values~%" )
    (format t "~{~15,4e~^ ~}~%" (coerce uerrbd 'list))
    (format t "~%Error estimates for the right singular values~%" )
    (format t "~{~15,4e~^ ~}~%" (coerce verrbd 'list)))

(defun test-dgesdd ()
  ;;

```

```

;; Matrix A:
;;   2.27   0.28  -0.48   1.07  -2.35   0.62
;;  -1.54  -1.67  -3.09   1.22   2.93  -7.39
;;   1.15   0.94   0.99   0.79  -1.45   1.03
;;  -1.94  -0.78  -0.21   0.63   2.30  -2.57
(let* ((m 4) ; rows
      (n 6) ; cols
      (a-mat (make-array (* m n) :element-type 'double-float
                          :initial-contents '(2.27d0 -1.54d0 1.15d0 -1.94d0
0.28d0 -1.67d0 0.94d0 -0.78d0
-0.48d0 -3.09d0 0.99d0 -0.21d0
1.07d0 1.22d0 0.79d0 0.63d0
-2.35d0 2.93d0 -1.45d0 2.30d0
0.62d0 -7.39d0 1.03d0 -2.57d0))))
      (s (make-array (min m n) :element-type 'double-float))
      (u (make-array (* m (min m n)) :element-type 'double-float))
      (vt (make-array (* n n) :element-type 'double-float))
      (lwork 1000)
      (work (make-array lwork :element-type 'double-float))
      (iwork (make-array (* 8 (min m n)) :element-type 'f2cl-lib:integer4)))
      (multiple-value-bind (z-jobz z-m z-n z-a z-lda z-s z-u z-ldu z-vt z-ldvt
                           z-work z-lwork z-iwork info)
        (dgesdd "Overwrite A by transpose(V)"
                 m n a-mat m s u m vt n work lwork iwork 0)
        (declare (ignore z-jobz z-m z-n z-a z-lda z-s z-u z-ldu z-vt z-ldvt
                        z-work z-lwork z-iwork ))
        ;; Display solution
        (cond ((zerop info)
              (print-dgesdd-results m n s u a-mat))
              (t
               (format t "Failure in DGESDD. Info = ~D~%" info)))
        (format t "Optimum workspace required = ~D~%" (truncate (aref work 0)))
        (format t "Workspace provided = ~D~%" lwork))))

;; Expected results (from
;;   http://www.nag.co.uk/lapack-ex/examples/results/dgesvd-ex.r)
;; DGESVD Example Program Results
;;
;; Singular values
;;   9.9966  3.6831  1.3569  0.5000
;; Left singular vectors (first n columns of U)
;;      1      2      3      4
;; 1 -0.2774 -0.6003 -0.1277  0.1323
;; 2 -0.2020 -0.0301  0.2805  0.7034
;; 3 -0.2918  0.3348  0.6453  0.1906
;; 4  0.0938 -0.3699  0.6781 -0.5399
;; 5  0.4213  0.5266  0.0413 -0.0575
;; 6 -0.7816  0.3353 -0.1645 -0.3957
;;
;; Right singular vectors by row (V**T)

```

```

;;          1          2          3          4
;;  1  -0.1921  0.8794 -0.2140  0.3795
;;  2  -0.8030 -0.3926 -0.2980  0.3351
;;  3   0.0041 -0.0752  0.7827  0.6178
;;  4  -0.5642  0.2587  0.5027 -0.6017
;;
;; Error estimate for the singular values
;;      1.1E-15
;;
;; Error estimates for the left singular vectors
;;      1.8E-16   4.8E-16   1.3E-15   2.2E-15
;;
;; Error estimates for the right singular vectors
;;      1.8E-16   4.8E-16   1.3E-15   1.3E-15
;;
(defun print-dgesvd-results (m n s vt a)
  (format t "~2%DGESVD Example Program Results~%" )
  (format t "Singular values~%" )
  (dotimes (k n)
    (format t "~20,14e" (aref s k)))
  (format t "~2%Left singular vectors~%" )
  (dotimes (r m)
    (dotimes (c n)
      (format t "~16,7e" (aref a (+ r (* c m))))))
    (terpri))
  (format t "~%Right singular vectors (first m rows of V**T)~%" )
  (dotimes (r n)
    (dotimes (c n)
      (format t "~16,7e" (aref vt (+ r (* c n))))))
    (terpri))
  ;; Compute error estimates for the singular vectors
  (let ((serrbd (* (aref s 0) (dlamch "Eps"))))
    (rcondv (make-array n :element-type 'double-float))
    (rcondv (make-array n :element-type 'double-float))
    (uerrbd (make-array n :element-type 'double-float))
    (verrbd (make-array n :element-type 'double-float)))
    (ddisna "Left" m n s rcondv 0)
    (ddisna "Right" m n s rcondv 0)
    (dotimes (k n)
      (setf (aref uerrbd k) (/ serrbd (aref rcondv k)))
      (setf (aref verrbd k) (/ serrbd (aref rcondv k))))
    (format t "Error estimate for the singular values~%" )
    (format t "~20,15g~%" serrbd)
    (format t "~%Error estimates for the left singular values~%" )
    (format t "~{~15,4e~^ ~}~%" (coerce uerrbd 'list))
    (format t "~%Error estimates for the right singular values~%" )
    (format t "~{~15,4e~^ ~}~%" (coerce verrbd 'list)))

(defun test-dgesvd ()
  ;;

```

```

;; Matrix A:
;;      2.27  -1.54   1.15  -1.94
;;      0.28  -1.67   0.94  -0.78
;;     -0.48  -3.09   0.99  -0.21
;;      1.07   1.22   0.79   0.63
;;     -2.35   2.93  -1.45   2.30
;;      0.62  -7.39   1.03  -2.57
(let* ((m 6) ; rows
      (n 4) ; cols
      (a-mat (make-array (* m n) :element-type 'double-float
        :initial-contents '(2.27d0 0.28d0 -0.48d0 1.07d0 -2.35d0 0.62d0
        -1.54d0 -1.67d0 -3.09d0 1.22d0 2.93d0 -7.39d0
        1.15d0 0.94d0 0.99d0 0.79d0 -1.45d0 1.03d0
        -1.94d0 -0.78d0 -0.21d0 0.63d0 2.30d0 -2.57d0)))
      (s (make-array (min m n) :element-type 'double-float))
      (u (make-array (* m (min m n)) :element-type 'double-float))
      (vt (make-array (* n n) :element-type 'double-float))
      (lwork (+ 10 (* 4 8)
        (* 64 (+ 10 8))))
      (work (make-array lwork :element-type 'double-float)))
  (multiple-value-bind (z-jobz z-jobvt z-m z-n z-a z-lda z-s z-u z-ldu z-vt
    z-ldvt z-work z-lwork info)
    (dgesvd "Overwrite A by U" "Singular vectors (V)"
      m n a-mat m s u m vt n work lwork 0)
    (declare (ignore z-jobz z-jobvt z-m z-n z-a z-lda z-s z-u z-ldu z-vt
      z-ldvt z-work z-lwork))
    ;; Display solution
    (cond ((zerop info)
      (print-dgesvd-results m n s vt a-mat))
      (t
        (format t "Failure in DGESDD. Info = ~D~%" info)))
    (format t "Optimum workspace required = ~D~%" (truncate (aref work 0)))
    (format t "Workspace provided = ~D~%" lwork))))

(defun make-complex-eigvec (n vr)
  (make-array (list n n)
    :displaced-to vr
    :element-type (array-element-type vr)))

(defun print-zgeev-results (e-val e-vec)
  (format t "~2%ZGEEV Example Program Results~%"
    (let ((n (length e-val)))
      (dotimes (k n)
        (format t "Eigenvalue(~D) = ~A~%" k (aref e-val k))
        (format t "~%Eigenvector(~D)~%" k)
        (dotimes (row n)
          (format t "~A~%" (aref e-vec row k)))
        (terpri))))

(defun test-zgeev ()

```

```

;; The matrix is
;;
;; #c(-3.97, -5.04) #c(-4.11, 3.70) #c(-0.34, 1.01) #c( 1.29, -0.86)
;; #c( 0.34, -1.50) #c( 1.52, -0.43) #c( 1.88, -5.38) #c( 3.36, 0.65)
;; #c( 3.31, -3.85) #c( 2.50, 3.45) #c( 0.88, -1.08) #c( 0.64, -1.48)
;; #c(-1.10, 0.82) #c( 1.81, -1.59) #c( 3.25, 1.33) #c( 1.57, -3.44)
;;
;; Recall that Fortran arrays are column-major order!
(let* ((n 4)
      (a-mat (make-array (* n n)
                          :element-type '(complex double-float)
                          :initial-contents '(#c(-3.97d0 -5.04d0)
#c( 0.34d0 -1.50d0)
#c( 3.31d0 -3.85d0)
#c(-1.10d0 0.82d0)
#c(-4.11d0 3.70d0)
#c( 1.52d0 -0.43d0)
#c( 2.50d0 3.45d0)
#c( 1.81d0 -1.59d0)
#c(-0.34d0 1.01d0)
#c( 1.88d0 -5.38d0)
#c( 0.88d0 -1.08d0)
#c( 3.25d0 1.33d0)
#c( 1.29d0 -0.86d0)
#c( 3.36d0 0.65d0)
#c( 0.64d0 -1.48d0)
#c( 1.57d0 -3.44d0))))))
      (lwork 660)
      (w (make-array n :element-type '(complex double-float)))
      (rw (make-array lwork :element-type 'double-float))
      (vl (make-array 0 :element-type '(complex double-float)))
      (vr (make-array (* n n) :element-type '(complex double-float)))
      (work (make-array lwork :element-type '(complex double-float))))
      (multiple-value-bind (z-jobvl z-jobvr z-n z-a z-lda z-w z-vl z-ldvl z-vr
z-ldvr z-work z-lwork z-rwork info)
        (zgeev "N" "V" n a-mat n w vl n vr n work lwork rw 0)
        (declare (ignore z-jobvl z-jobvr z-n z-a z-lda z-w z-vl z-ldvl z-vr
z-ldvr z-work z-lwork z-rwork))
        ;; Display solution
        (cond ((zerop info)
              (print-zgeev-results w
              (make-complex-eigvec n vr))))
        (t
         (format t "Failure in DGEEV. INFO = ~D~%" info)))
        ;; Display workspace info
        (format t "Optimum workspace required = ~D~%"
              (truncate (realpart (aref work 0))))
        (format t "Workspace provided = ~D~%" lwork))))))

(defun do-all-lapack-tests ()

```

```

(test-dgeev)
(test-dgeevx)
(test-dgesv)
(test-dgesdd)
(test-dgesvd)
(test-zgeev))

;;; $Log$
;;; Revision 1.11  2006/12/01 04:29:29  rtoy
;;; Create packages for BLAS and LAPACK routines.
;;;
;;; blas.system:
;;; o Converted files are in the BLAS package.
;;; o Add blas-package defsystem to load the package definition.
;;;
;;; lapack.system:
;;; o Converted files are in the LAPACK package.
;;; o Add lapack-package defsystem to load the package definition.
;;;
;;; lapack/lapack-tests.lisp:
;;; o Tests are in the LAPACK package
;;;
;;; Revision 1.10  2006/11/28 15:49:01  rtoy
;;; Print out short title for each test.
;;;
;;; Revision 1.9  2006/11/27 22:22:23  rtoy
;;; Add expected results.
;;;
;;; Revision 1.8  2006/11/27 20:04:33  rtoy
;;; Add DGESVD and update files and tests appropriately.
;;;
;;; Revision 1.7  2006/11/27 15:23:29  rtoy
;;; Add function to run all the tests.
;;;
;;; Revision 1.6  2006/11/26 23:26:47  rtoy
;;; packages/lapack.system:
;;; o Add DGESDD and dependencies
;;; o Add DDISNA to compute condition number of singular vectors
;;;
;;; packages/lapack/.cvsignore:
;;; o Ignore new generated Lisp files.
;;;
;;; packages/lapack/lapack-tests.lisp:
;;; o Add test for DGESDD
;;;
;;; Revision 1.5  2006/11/26 14:26:42  rtoy
;;; Add expected results for DGESV.
;;;
;;; Revision 1.4  2006/11/26 14:24:46  rtoy
;;; packages/lapack.system:

```



```

;;; o DGEV and dependencies
;;;
;;; packages/.cvsignore:
;;; o Ignore generated dgesv.lisp and dependencies
;;;
;;; packages/lapack/lapack-tests.lisp:
;;; o Test routine for DGEV
;;;
;;; Revision 1.3 2006/11/26 05:31:16 rtoy
;;; packages/lapack.system:
;;; o Add DGEV and dependencies
;;;
;;; packages/lapack/lapack-tests.lisp:
;;; o Add test for DGEV
;;; o Add comments
;;;
;;; packages/lapack/dgeev.f:
;;; packages/lapack/dlacon.f:
;;; packages/lapack/dlaexc.f:
;;; packages/lapack/dlaqtr.f:
;;; packages/lapack/dlasy2.f:
;;; packages/lapack/dtrexc.f:
;;; packages/lapack/dtrsna.f:
;;; o New files for DGEV and dependencies.
;;;
;;; Revision 1.2 2006/11/26 04:53:22 rtoy
;;; Add comments
;;;
;;; Revision 1.1 2006/11/26 04:51:05 rtoy
;;; packages/lapack.system:
;;; o Add defsystem for LAPACK tests
;;;
;;; packages/lapack/lapack-tests.lisp:
;;; o Add simple tests for LAPACK. (Currently only DGEV).
;;;

```


Chapter 8

Chunk collections

— BLAS FORTRAN —

```
\begin{chunk}{dcabs1.f}  
\begin{chunk}{lsame.f}  
\begin{chunk}{xerbla.f}  
\begin{chunk}{dasum.f}  
\begin{chunk}{daxpy.f}  
\begin{chunk}{dcopy.f}  
\begin{chunk}{ddot.f}  
\begin{chunk}{dnrm2.f}  
\begin{chunk}{drotg.f}  
\begin{chunk}{drot.f}  
\begin{chunk}{dscal.f}  
\begin{chunk}{dswap.f}  
\begin{chunk}{dzasum.f}  
\begin{chunk}{dznrm2.f}  
\begin{chunk}{icamax.f}  
\begin{chunk}{idamax.f}  
\begin{chunk}{isamax.f}  
\begin{chunk}{izamax.f}  
\begin{chunk}{zaxpy.f}  
\begin{chunk}{zcopy.f}  
\begin{chunk}{zdotc.f}  
\begin{chunk}{zdotu.f}  
\begin{chunk}{zdscal.f}  
\begin{chunk}{zrotg.f}  
\begin{chunk}{zscal.f}  
\begin{chunk}{zswap.f}  
\begin{chunk}{dgbmv.f}  
\begin{chunk}{dgemv.f}  
\begin{chunk}{dger.f}
```

```

\begin{chunk}{dsbmv.f}
\begin{chunk}{dspmv.f}
\begin{chunk}{dspr2.f}
\begin{chunk}{dspr.f}
\begin{chunk}{dsymv.f}
\begin{chunk}{dsyr2.f}
\begin{chunk}{dsyr.f}
\begin{chunk}{dtbmv.f}
\begin{chunk}{dtbsv.f}
\begin{chunk}{dtpmv.f}
\begin{chunk}{dtpsv.f}
\begin{chunk}{dtrmv.f}
\begin{chunk}{dtrsv.f}
\begin{chunk}{zgbmv.f}
\begin{chunk}{zgemv.f}
\begin{chunk}{zgerc.f}
\begin{chunk}{zgeru.f}
\begin{chunk}{zhbmv.f}
\begin{chunk}{zhemv.f}
\begin{chunk}{zher2.f}
\begin{chunk}{zher.f}
\begin{chunk}{zhpmv.f}
\begin{chunk}{zhpr2.f}
\begin{chunk}{zhpr.f}
\begin{chunk}{ztbmv.f}
\begin{chunk}{ztbsv.f}
\begin{chunk}{ztpmv.f}
\begin{chunk}{ztpsv.f}
\begin{chunk}{ztrmv.f}
\begin{chunk}{ztrsv.f}
\begin{chunk}{dgemm.f}
\begin{chunk}{dsymm.f}
\begin{chunk}{dsyr2k.f}
\begin{chunk}{dsyrk.f}
\begin{chunk}{dtrmm.f}
\begin{chunk}{dtrsm.f}
\begin{chunk}{zgemm.f}
\begin{chunk}{zhemm.f}
\begin{chunk}{zher2k.f}
\begin{chunk}{zherk.f}
\begin{chunk}{zsymm.f}
\begin{chunk}{zsyr2k.f}
\begin{chunk}{zsyrk.f}
\begin{chunk}{ztrmm.f}
\begin{chunk}{ztrsm.f}

```

```

\begin{chunk}{dbdsdc.f}
\begin{chunk}{dbdsqr.f}
\begin{chunk}{ddisna.f}
\begin{chunk}{dgebak.f}
\begin{chunk}{dgebal.f}
\begin{chunk}{dgebd2.f}
\begin{chunk}{dgebrd.f}
\begin{chunk}{dgeev.f}
\begin{chunk}{dgeevx.f}
\begin{chunk}{dgehd2.f}
\begin{chunk}{dgehrd.f}
\begin{chunk}{dgelq2.f}
\begin{chunk}{dgelqf.f}
\begin{chunk}{dgeqr2.f}
\begin{chunk}{dgeqrf.f}
\begin{chunk}{dgesdd.f}
\begin{chunk}{dgesvd.f}
\begin{chunk}{dgesv.f}
\begin{chunk}{dgetf2.f}
\begin{chunk}{dgetrf.f}
\begin{chunk}{dgetrs.f}
\begin{chunk}{dhseqr.f}
\begin{chunk}{disnan.f}
\begin{chunk}{dlabad.f}
\begin{chunk}{dlabrd.f}
\begin{chunk}{dlacon.f}
\begin{chunk}{dlacpy.f}
\begin{chunk}{dladiv.f}
\begin{chunk}{dlaed6.f}
\begin{chunk}{dlaexc.f}
\begin{chunk}{dlaqrf.f}
\begin{chunk}{dlahrd.f}
\begin{chunk}{dlaisnan.f}
\begin{chunk}{dlaln2.f}
\begin{chunk}{dlamch.f}
\begin{chunk}{dlamc1.f}
\begin{chunk}{dlamc2.f}
\begin{chunk}{dlamc3.f}
\begin{chunk}{dlamc4.f}
\begin{chunk}{dlamc5.f}
\begin{chunk}{dlamrg.f}
\begin{chunk}{dlange.f}
\begin{chunk}{dlanhs.f}
\begin{chunk}{dlanst.f}
\begin{chunk}{dlanv2.f}
\begin{chunk}{dlapy2.f}
\begin{chunk}{dlapy3.f}
\begin{chunk}{dlaqtr.f}
\begin{chunk}{dlarfb.f}
\begin{chunk}{dlarfg.f}

```

```

\begin{chunk}{dlarf.f}
\begin{chunk}{dlarft.f}
\begin{chunk}{dlarfx.f}
\begin{chunk}{dlartg.f}
\begin{chunk}{dlas2.f}
\begin{chunk}{dlascl.f}
\begin{chunk}{dlasd0.f}
\begin{chunk}{dlasd1.f}
\begin{chunk}{dlasd2.f}
\begin{chunk}{dlasd3.f}
\begin{chunk}{dlasd4.f}
\begin{chunk}{dlasd5.f}
\begin{chunk}{dlasd6.f}
\begin{chunk}{dlasd7.f}
\begin{chunk}{dlasd8.f}
\begin{chunk}{dlasda.f}
\begin{chunk}{dlasdq.f}
\begin{chunk}{dlasdt.f}
\begin{chunk}{dlaset.f}
\begin{chunk}{dlasq1.f}
\begin{chunk}{dlasq2.f}
\begin{chunk}{dlasq3.f}
\begin{chunk}{dlasq4.f}
\begin{chunk}{dlasq5.f}
\begin{chunk}{dlasq6.f}
\begin{chunk}{dlasr.f}
\begin{chunk}{dlasrt.f}
\begin{chunk}{dlassq.f}
\begin{chunk}{dlasv2.f}
\begin{chunk}{dlaswp.f}
\begin{chunk}{dlasy2.f}
\begin{chunk}{dorg2r.f}
\begin{chunk}{dorgbr.f}
\begin{chunk}{dorghr.f}
\begin{chunk}{dorgl2.f}
\begin{chunk}{dorglq.f}
\begin{chunk}{dorgqr.f}
\begin{chunk}{dorm2r.f}
\begin{chunk}{dormbr.f}
\begin{chunk}{dorml2.f}
\begin{chunk}{dormlq.f}
\begin{chunk}{dormqr.f}
\begin{chunk}{dtrevc.f}
\begin{chunk}{dtrexc.f}
\begin{chunk}{dtrsna.f}
\begin{chunk}{ieeack.f}
\begin{chunk}{ilaenv.f}
\begin{chunk}{ilazlc.f}
\begin{chunk}{ilazlr.f}
\begin{chunk}{zgebak.f}

```

```

\begin{chunk}{zgebal.f}
\begin{chunk}{zgeev.f}
\begin{chunk}{zgehd2.f}
\begin{chunk}{zgehrd.f}
\begin{chunk}{zhseqr.f}
\begin{chunk}{zlacgv.f}
\begin{chunk}{zlacpy.f}
\begin{chunk}{zladiw.f}
\begin{chunk}{zlahqr.f}
\begin{chunk}{zlahr2.f}
\begin{chunk}{zlange.f}
\begin{chunk}{zlaqr0.f}
\begin{chunk}{zlaqr1.f}
\begin{chunk}{zlaqr2.f}
\begin{chunk}{zlaqr3.f}
\begin{chunk}{zlaqr4.f}
\begin{chunk}{zlaqr5.f}
\begin{chunk}{zlarfb.f}
\begin{chunk}{zlarf.f}
\begin{chunk}{zlzrfg.f}
\begin{chunk}{zlarft.f}
\begin{chunk}{zlartg.f}
\begin{chunk}{zlascl.f}
\begin{chunk}{zlasel.f}
\begin{chunk}{zlassq.f}
\begin{chunk}{zlatrs.f}
\begin{chunk}{zrot.f}
\begin{chunk}{ztrevc.f}
\begin{chunk}{ztrexcl.f}
\begin{chunk}{zung2r.f}
\begin{chunk}{zungqr.f}
\begin{chunk}{zunm2r.f}
\begin{chunk}{zunmhr.f}
\begin{chunk}{zunmqr.f}

```

— Numerics —

```
(in-package "BOOT")
```

```

\getchunk{BLAS 1 dcabs1}
\getchunk{BLAS 1 dasum}
\getchunk{BLAS 1 daxpy}
\getchunk{BLAS 1 dcopy}
\getchunk{BLAS 1 ddot}
\getchunk{BLAS 1 dnorm2}
\getchunk{BLAS 1 drotg}

```

```

\getchunk{BLAS 1 drot}
\getchunk{BLAS 1 dscal}
\getchunk{BLAS 1 dswap}
\getchunk{BLAS 1 dzasum}
\getchunk{BLAS 1 dznrm2}
\getchunk{BLAS 1 icamax}
\getchunk{BLAS 1 idamax}
\getchunk{BLAS 1 isamax}
\getchunk{BLAS 1 izamax}
\getchunk{BLAS 1 zaxpy}

```

— untested —

```

\getchunk{BLAS lsame}
\getchunk{BLAS xerbla}

\getchunk{BLAS 1 zcopy}
\getchunk{BLAS 1 zdotc}
\getchunk{BLAS 1 zdotu}
\getchunk{BLAS 1 zdscal}
\getchunk{BLAS 1 zrotg}
\getchunk{BLAS 1 zscal}
\getchunk{BLAS 1 zswap}

\getchunk{BLAS 2 dgbmv}
\getchunk{BLAS 2 dgemv}
\getchunk{BLAS 2 dger}
\getchunk{BLAS 2 dsbmv}
\getchunk{BLAS 2 dspmv}
\getchunk{BLAS 2 dspr2}
\getchunk{BLAS 2 dspr}
\getchunk{BLAS 2 dsymv}
\getchunk{BLAS 2 dsyr2}
\getchunk{BLAS 2 dsyr}
\getchunk{BLAS 2 dtbmv}
\getchunk{BLAS 2 dtbsv}
\getchunk{BLAS 2 dtpmv}
\getchunk{BLAS 2 dtpsv}
\getchunk{BLAS 2 dtrmv}
\getchunk{BLAS 2 dtrsv}
\getchunk{BLAS 2 zgbbmv}
\getchunk{BLAS 2 zgemv}
\getchunk{BLAS 2 zgerc}
\getchunk{BLAS 2 zgeru}
\getchunk{BLAS 2 zhbmv}
\getchunk{BLAS 2 zhenv}
\getchunk{BLAS 2 zher2}

```



```

\getchunk{BLAS 2 zher}
\getchunk{BLAS 2 zhpmv}
\getchunk{BLAS 2 zhpr2}
\getchunk{BLAS 2 zhpr}
\getchunk{BLAS 2 ztbmv}
\getchunk{BLAS 2 ztbsv}
\getchunk{BLAS 2 ztpmv}
\getchunk{BLAS 2 ztpsv}
\getchunk{BLAS 2 ztrmv}
\getchunk{BLAS 2 ztrsv}

\getchunk{BLAS 3 dgemm}
\getchunk{BLAS 3 dsymm}
\getchunk{BLAS 3 dsyr2k}
\getchunk{BLAS 3 dsyrk}
\getchunk{BLAS 3 dtrmm}
\getchunk{BLAS 3 dtrsm}
\getchunk{BLAS 3 zgemm}
\getchunk{BLAS 3 zhemm}
\getchunk{BLAS 3 zher2k}
\getchunk{BLAS 3 zherk}
\getchunk{BLAS 3 zsymm}
\getchunk{BLAS 3 zsyr2k}
\getchunk{BLAS 3 zsyrk}
\getchunk{BLAS 3 ztrmm}
\getchunk{BLAS 3 ztrsm}

\getchunk{LAPACK dbdsdc}
\getchunk{LAPACK dbdsqr}
\getchunk{LAPACK ddisna}
\getchunk{LAPACK dgebak}
\getchunk{LAPACK dgebal}
\getchunk{LAPACK dgebd2}
\getchunk{LAPACK dgebrd}
\getchunk{LAPACK dgeev}
\getchunk{LAPACK dgeevx}
\getchunk{LAPACK dgehd2}
\getchunk{LAPACK dgehrd}
\getchunk{LAPACK dgelq2}
\getchunk{LAPACK dgelqf}
\getchunk{LAPACK dgeqr2}
\getchunk{LAPACK dgeqrf}
\getchunk{LAPACK dgesdd}
\getchunk{LAPACK dgesvd}
\getchunk{LAPACK dgesv}
\getchunk{LAPACK dgetf2}
\getchunk{LAPACK dgetrf}
\getchunk{LAPACK dgetrs}
\getchunk{LAPACK dhseqr}
\getchunk{LAPACK disnan}

```

```
\getchunk{LAPACK dlabad}  
\getchunk{LAPACK dlabrd}  
\getchunk{LAPACK dlacon}  
\getchunk{LAPACK dlacpy}  
\getchunk{LAPACK dladiv}  
\getchunk{LAPACK dlaed6}  
\getchunk{LAPACK dlaexc}  
\getchunk{LAPACK dlahqr}  
\getchunk{LAPACK dlahrd}  
\getchunk{LAPACK dlaisnan}  
\getchunk{LAPACK dlaln2}  
\getchunk{LAPACK dlamch}  
\getchunk{LAPACK dlamc1}  
\getchunk{LAPACK dlamc2}  
\getchunk{LAPACK dlamc3}  
\getchunk{LAPACK dlamc4}  
\getchunk{LAPACK dlamc5}  
\getchunk{LAPACK dlamrg}  
\getchunk{LAPACK dlange}  
\getchunk{LAPACK dlanhs}  
\getchunk{LAPACK dlanst}  
\getchunk{LAPACK dlanv2}  
\getchunk{LAPACK dlapy2}  
\getchunk{LAPACK dlapy3}  
\getchunk{LAPACK dlaqtr}  
\getchunk{LAPACK dlarfb}  
\getchunk{LAPACK dlarfg}  
\getchunk{LAPACK dlarf}  
\getchunk{LAPACK dlarft}  
\getchunk{LAPACK dlarfx}  
\getchunk{LAPACK dlartg}  
\getchunk{LAPACK dlas2}  
\getchunk{LAPACK dlascl}  
\getchunk{LAPACK dlasd0}  
\getchunk{LAPACK dlasd1}  
\getchunk{LAPACK dlasd2}  
\getchunk{LAPACK dlasd3}  
\getchunk{LAPACK dlasd4}  
\getchunk{LAPACK dlasd5}  
\getchunk{LAPACK dlasd6}  
\getchunk{LAPACK dlasd7}  
\getchunk{LAPACK dlasd8}  
\getchunk{LAPACK dlasda}  
\getchunk{LAPACK dlasdq}  
\getchunk{LAPACK dlasdt}  
\getchunk{LAPACK dlaset}  
\getchunk{LAPACK dlasq1}  
\getchunk{LAPACK dlasq2}  
\getchunk{LAPACK dlasq3}  
\getchunk{LAPACK dlasq4}
```

```
\getchunk{LAPACK dlasq5}  
\getchunk{LAPACK dlasq6}  
\getchunk{LAPACK dlasr}  
\getchunk{LAPACK dlasrt}  
\getchunk{LAPACK dlassq}  
\getchunk{LAPACK dlasv2}  
\getchunk{LAPACK dlaswp}  
\getchunk{LAPACK dlasy2}  
\getchunk{LAPACK dorg2r}  
\getchunk{LAPACK dorgbr}  
\getchunk{LAPACK dorghr}  
\getchunk{LAPACK dorgl2}  
\getchunk{LAPACK dorglq}  
\getchunk{LAPACK dorgqr}  
\getchunk{LAPACK dorm2r}  
\getchunk{LAPACK dormbr}  
\getchunk{LAPACK dorml2}  
\getchunk{LAPACK dormlq}  
\getchunk{LAPACK dormqr}  
\getchunk{LAPACK dtrevc}  
\getchunk{LAPACK dtrexc}  
\getchunk{LAPACK dtrsna}  
\getchunk{LAPACK ieeec}  
\getchunk{LAPACK ilaenv}  
\getchunk{LAPACK ilazlc}  
\getchunk{LAPACK ilazlr}  
\getchunk{LAPACK zgebak}  
\getchunk{LAPACK zgebal}  
\getchunk{LAPACK zgeev}  
\getchunk{LAPACK zgehd2}  
\getchunk{LAPACK zgehrd}  
\getchunk{LAPACK zhseqr}  
\getchunk{LAPACK zlacgv}  
\getchunk{LAPACK zlapcy}  
\getchunk{LAPACK zladiv}  
\getchunk{LAPACK zlahqr}  
\getchunk{LAPACK zlahr2}  
\getchunk{LAPACK zlange}  
\getchunk{LAPACK zlaqr0}  
\getchunk{LAPACK zlaqr1}  
\getchunk{LAPACK zlaqr2}  
\getchunk{LAPACK zlaqr3}  
\getchunk{LAPACK zlaqr4}  
\getchunk{LAPACK zlaqr5}  
\getchunk{LAPACK zlarfb}  
\getchunk{LAPACK zlarf}  
\getchunk{LAPACK zlarfg}  
\getchunk{LAPACK zlarft}  
\getchunk{LAPACK zlartg}  
\getchunk{LAPACK zlascl}
```

```
\getchunk{LAPACK zaset}  
\getchunk{LAPACK zlassq}  
\getchunk{LAPACK zlatrs}  
\getchunk{LAPACK zrot}  
\getchunk{LAPACK ztrevc}  
\getchunk{LAPACK ztrexc}  
\getchunk{LAPACK zung2r}  
\getchunk{LAPACK zunghr}  
\getchunk{LAPACK zungqr}  
\getchunk{LAPACK zunm2r}  
\getchunk{LAPACK zunmhr}  
\getchunk{LAPACK zunmqr}
```

Bibliography

- [1] documentation source <ftp://ftp.netlib.org/lapack/manpages.tgz>
- [2] documentation source <http://www.math.utah.edu/software/lapack/lapack-blas.html>
- [3] documentation source Written on 22-October-1986. Jack Dongarra, Argonne National Lab. Jeremy Du Croz, Nag Central Office. Sven Hammarling, Nag Central Office. Richard Hanson, Sandia National Labs.
- [4] Householder, Alston S. “Principles of Numerical Analysis” Dover Publications, Mineola, NY ISBN 0-486-45312-X (1981)
- [5] Golub, Gene H. and Van Loan, Charles F. “Matrix Computations” Johns Hopkins University Press ISBN 0-8018-3772-3 (1989)
- [6] Higham, Nicholas J. “Accuracy and stability of numerical algorithms” SIAM Philadelphia, PA ISBN 0-89871-521-0 (2002)

Chapter 9

Index

Index

- BLAS1, [42](#)
- BlasLevelOne, [42](#)
- char-equal
 - calledby dbdsdc, [941](#)
- dbdsdc
 - calls char-equal, [941](#)
 - calls dcopy, [941](#)
 - calls dlamch, [941](#)
 - calls dlanst, [941](#)
 - calls dlartg, [941](#)
 - calls dlascl, [941](#)
 - calls dlasd0, [941](#)
 - calls dlasda, [941](#)
 - calls dlasdq, [941](#)
 - calls dlaset, [941](#)
 - calls dlasr, [941](#)
 - calls dswap, [941](#)
 - calls ilaenv, [941](#)
 - calls xerbla, [941](#)
- dcopy
 - calledby dbdsdc, [941](#)
- dlamch
 - calledby dbdsdc, [941](#)
- dlanst
 - calledby dbdsdc, [941](#)
- dlartg
 - calledby dbdsdc, [941](#)
- dlascl
 - calledby dbdsdc, [941](#)
- dlasd0
 - calledby dbdsdc, [941](#)
- dlasda
 - calledby dbdsdc, [941](#)
- dlasdq
 - calledby dbdsdc, [941](#)
- dlaset
 - calledby dbdsdc, [941](#)
- dlasr
 - calledby dbdsdc, [941](#)
- dswap
 - calledby dbdsdc, [941](#)
- ilaenv
 - calledby dbdsdc, [941](#)
- xerbla
 - calledby dbdsdc, [941](#)